UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Utilización de servicios Cloud con Interfaces REST

Autores: Albarrazín, Pablo José – Legajo: 9096/8

Director: Prof. Tinetti, Fernando G.

Carrera: Licenciatura en Informática – Plan 2003/07

Resumen

La evolución en las ciencias informáticas nos ha demostrado la importancia de comprender la información que manejamos día a día. Hoy no solo es importante contar con los datos correctos, sino que también se consideran críticos al mecanismo de procesamiento que se necesita para interpretarlos, y al tiempo que demande dicho procesamiento.
Este trabajo tiene como objetivo la investigación de diferentes plataformas en la nube que permitan la ejecución de servicios. Se hará foco en servicios que sean capaces de ser ejecutados bajo interfaces REST, y que con esto faciliten la administración de su ejecución y consulta, entre otras operaciones.
Como ejemplo primario, se espera también demostrar la posibilidad de implementar la ejecución de procesos de Map Reduce vía REST, alimentando a dicho proceso mediante múltiples fuentes.

Palabras Claves

HTTP, REST, Hadoop, HDFS, YARN, Sistema de archivos, MapReduce, Big Data, Cloud Computing, Windows Azure, API

Conclusiones

Los desarrollos presentados han demostrado el objetivo del presente trabajo: lograr la ejecución de servicios en ambientes clusterizados bajo interfaces REST, alimentando dicho servicio a través de múltiples fuentes de datos. Se logra, además, una interfaz de abstracción para interactuar con el software de Hadoop.

Trabajos Realizados

- Desarrollo en Java para la gestión de archivos y carpetas en el sistema de archivos distribuido de Hadoop – HDFS, a través de una interfaz HTTP REST

- Desarrollo en Java para la gestión de trabajos YARN en Hadoop a través de una interfaz HTTP REST

Trabajos Futuros

- Realizar un análisis de la performance de las herramientas desarrolladas (speed-up, eficiencia, escalabilidad, entre otras)
- Generar automatización en el proceso de instalación y configuración de Apache Hadoop dentro de un clúster
- Estandarizar el proceso de empaquetado y despliegue de las aplicaciones desarrolladas.
Documentar los desarrollos realizados a través de alguna herramienta para representar APIs que son RESTful

Fecha de la presentación: Noviembre 2016

Universidad Nacional de La Plata

Facultad de Informática

Utilización de servicios Cloud con Interfaces REST

Tesina de Licenciatura en Informática

Autor: Albarrazín, Pablo J.

Director: Tinetti, Fernando G.

Fecha: Noviembre 2016

Índice general

Índice de figuras

Capítulo 1 - INTRODUCCIÓN

La evolución en las ciencias informáticas nos ha demostrado la importancia de comprender la información que manejamos día a día. Hoy no solo es importante contar con los datos correctos, sino que también se consideran críticos al mecanismo de procesamiento que se necesita para interpretarlos, y al tiempo que demande dicho procesamiento.

La última década se ha visto marcada por un incremento considerable en la cantidad de datos que una persona produce diariamente [1]. Desde sus gustos, preferencias, movimientos financieros, entre otros, hacen al perfil de un individuo.

El problema al que se suelen enfrentar los analistas de datos entonces no es solamente a la obtención de los mismos, sino también a un rápido procesamiento y su consecuente obtención de conclusiones para una rápida y efectiva toma de decisiones, clave en campos como la medicina, el pronóstico o alertas de catástrofes climáticas, y negocios financieros, entre otros [2].

Bajo esta demanda creciente se produce el nacimiento y utilización masiva de términos como Big Data, Business Intelligence [3], NoSql [4], Data Analysis, entre otros.

Gracias al modelo de programación Map-Reduce [5], y a las nuevas tecnologías en la nube, como big data [6], es posible realizar un análisis de grandes volúmenes de datos en un tiempo aceptable. Sin embargo sigue pendiente la efectiva recolección de datos a través de las diferentes fuentes que puedan alimentar a un proceso determinado, como así también la fácil administración para el manejo de los procesos (ejecutarlos, conocer su estado, obtener su resultado, etc.).

Este trabajo tiene como objetivo la investigación de diferentes plataformas en la nube que permitan la ejecución de servicios. Se hará foco en servicios que sean capaces de ser ejecutados bajo interfaces REST [7], y que con esto faciliten la administración de su ejecución y consulta, entre otras operaciones.
Como ejemplo primario, se espera también demostrar la posibilidad de implementar la ejecución de procesos de Map Reduce vía REST, alimentando a dicho proceso mediante múltiples fuentes.

1.1 Estructura de la tesina

Lo que resta del documento se organiza de la siguiente manera:

- **Capítulo 2:** se define y explica en profundidad el concepto de REST, junto con ejemplos.
- **Capítulo 3:** en este capítulo se trabaja sobre el concepto de computación en la nube. Se realiza una definición de conceptos y se presentan ejemplos.
- **Capítulo 4:** se trabaja sobre el término “Big Data”, ofreciendo una definición para luego hacer foco en el modelo de programación Map Reduce.

- **Capítulo 5:** se hace un desarrollo de la herramienta Hadoop, explicando sus componentes y funcionamiento.
- **Capítulo 6:** se presentan las herramientas desarrolladas, explicando las tecnologías involucradas en su desarrollo. Se las muestra en funcionamiento haciendo un análisis sobre las ventajas de la utilización de las mismas.
- **Capítulo 7:** se presentan conclusiones sobre el trabajo presentado, dando lugar a líneas de trabajo futuro

Además de los capítulos presentados, se incluye un **Apéndice** que sirve de guía para configurar un clúster de Hadoop en un ambiente de computación en la nube. En este caso se introduce sobre el servicio en la nube de Microsoft, Azure.

Capítulo 2 - REST

El término REST (del inglés Representational State Transfer) surge por primera vez en la Universidad de California, Irvine, en el año 2000. Roy T. Fielding, en su tesis doctoral titulada "Architectural Styles and the Design of Network-based Software Architectures" [8], hace su propia definición, la cual se traduce como sigue:

> "REST se define para tener una imagen de cómo se comporta una aplicación Web bien diseñada: una red de "páginas" (es decir, una máquina de estados virtuales), donde el usuario avanza a través de la aplicación seleccionando links (transición de estados), lo que resulta en una "página siguiente" (representado por el siguiente estado de la aplicación) la cual se transfiere y renderiza al usuario para que éste la utilice"

En palabras más simples, REST es un conjunto de recomendaciones que forman un estilo o filosofía arquitectural, muy cercana a un paradigma. Es útil hacer esta aclaración, ya que en muchas ocasiones se lo suele confundir con un estándar. Sucede esto por la amplia aceptación que ha tenido desde su aparición en diversos ámbitos. Como anticipo de lo que viene, la World Wide Web es un ejemplo claro y funcionando de REST. Aunque no sea un estándar, REST promueve el uso de varias tecnologías estándar como HTTP (con quien más se la confunde), URI, XML, JSON, entre otras.

Este capítulo hará un recorrido por las principales características que componen a REST. Una vez entendido como funciona, se incluirán ejemplos, casos de éxitos reales y recientes de servicios de consumo masivo que estén implementando REST.

Se trata luego, un modelo propuesto por Leonard Richardson [9], el cual es básicamente una receta para llegar a construir un sistema que cumpla con REST de manera consistente.

Por último se ahondará con fuerza sobre una de las recomendaciones más fuertes de REST: Hipermedia. Se definirá y explicará con ejemplos reales qué es y por qué resulta tan importante para el desarrollo de sistemas que interactúen por la web.

2.1 Características

Tal como dijimos en la introducción, REST no es un estándar, sino más bien, un conjunto de recomendaciones para un estilo de arquitectura específica. Sobre ese conjunto de recomendaciones, existen algunas muy fuertes, que son esperadas en cualquier sistema REST que esté bien diseñado e implementado. Estas recomendaciones no sólo fueron propuestas por quien es considerado el padre del término REST, Roy Fielding, sino que fueron evolucionando y adaptándose en el tiempo, dando origen a nuevas recomendaciones que propuso la comunidad. Se nombran a continuación, las más relevantes:

- Cliente-Servidor: Uno de los puntos más fuertes dentro de REST corresponde a este estilo de arquitectura: cliente-y-servidor. De esta manera se logran separar responsabilidades. Por un lado tenemos las responsabilidades de la interfaz de usuario, y por otro lado, las responsabilidades que hacen al almacenamiento y tratamiento de

información. Con esto se cumplen dos objetivos: aumentar significativamente la portabilidad de la interfaz de usuario a través de múltiples plataformas, y al mismo tiempo mejorar y aumentar la escalabilidad simplificando los componentes en el servidor. La evolución de la parte cliente y servidor puede (y sucede muchas veces) darse de manera independiente.

- Sin estado: La comunicación entre cliente-y-servidor en un sistema REST es sin estado. Esto implica que cada petición que se hace desde el cliente hacia el servidor debe contener toda la información necesaria para que la petición sea entendida por el servidor. Esta decisión de diseño tiene un costo-beneficio, que en algunos escenarios puede interpretarse como una desventaja: la performance de la red puede verse afectada por el incremento del envío de datos repetidos, ya que como se dijo, los datos no pueden quedar almacenados en un contexto compartido dentro del servidor.
- Almacenamiento en cache: A lo que hemos explicado hasta ahora (sistema cliente-servidor que no almacena estado), se le puede sumar la posibilidad de almacenar en cache ciertas interacciones petición ←→ respuesta. Esto implica directamente una mejora al usuario que realiza la petición, ya que puede ver reducido significativamente el tiempo de respuesta. Para poder almacenar en cache, el servidor "etiqueta" a los datos como "disponibles para almacenar en cache" de alguna manera. Hay que tener especial cuidado con los datos que se guardan en la cache, ya que en ciertas ocasiones puede ocurrir que esos datos difieran significativamente de los datos que se hubiesen obtenido realizando la petición al servidor directamente. Existen diferentes mecanismos para detectar y evitar este tipo de inconsistencias, por ejemplo, el servidor podría indicar por cuánto tiempo es recomendable almacenar en cache los elementos. De esta manera se puede mejorar la confiabilidad del sistema de caché que pueda implementar el cliente. Un servidor que exponga datos disponibles para cache también incrementa su escalabilidad.
- Interfaz Uniforme: Una de las principales características de REST que lo diferencia de otras arquitecturas, es que ofrece una interfaz uniforme entre las diferentes componentes. Esto permite que no exista un acoplamiento entre las arquitecturas de las diferentes componentes. Yendo a una ventaja más importante, permite que cada componente evolucione, escale, y se desarrolle de manera independiente. Esta funcionalidad consta de cuatro restricciones de la interfaz:
    - Identificación de recursos: Toda información que pueda ser nombrada puede ser un recurso. Un recurso es la abstracción de información más significativa que provee REST. En su definición, un recurso "R" es una función de pertenencia que varía en el tiempo, Mr(t), donde en un tiempo "t" mapea a un conjunto de entidades. En el caso de la Web, la forma convencional de identificar un recurso es mediante una URI [10]. La URI por lo general no solo identifica a un recurso, sino que también permite que sea manipulable por un protocolo de capa de aplicación [11] como puede ser HTTP. Un "identificador de recurso" es muy importante ya que identifica a un recurso en particular que esté siendo parte de una interacción entre diferentes componentes.
    - Manipulación de recursos a través de representaciones: Los recursos que ya describimos deben ser accedidos, modificados e intercambiados entre diferentes componentes. Entenderse entre ellas es fundamental, y es lo que hoy

permite, por ejemplo, que un navegador web acceda, procese e interprete un sitio web para que sea visto por un usuario final. De esta manera podemos definir a una “representación” como una secuencia de bytes que llamamos “datos”, los cuales vienen acompañados además con información sobre esa secuencia, denominada “metadatos de la representación” que sirve para describir esos bytes y poder interpretarlos. Entre los formatos de representación más populares podemos nombre HTML, PNG, JSON, XML, sólo por nombrar algunos. Un servicio o componente tiene la posibilidad de exponer sus recursos a través de más de un tipo de representación. De esta manera se aumenta la interoperabilidad de sistemas, permitiendo mayor escalabilidad y compatibilidad. Un punto a favor que vale aclarar, es que un recurso, en sus diferentes representaciones, sigue siendo identificado de la misma manera.

- Mensajes auto-descriptivos: Esta restricción es bastante sencilla. Se establece que en cada petición se recibe una respuesta que contiene toda la información y meta-data necesaria para ser procesada, interpretada, y así se pueda completar la tarea. Esta restricción está fuertemente vinculada con una de las características más fuertes de REST, que establece que la comunicación entre las componentes es sin estado, y que el servidor en ningún momento tiene un contexto compartido entre todas las peticiones
- Hipermedia como el motor de estado de la aplicación: También conocida como *HATEOAS* [7], por su sigla en inglés: Hypermedia as the engine of application state, esta restricción viene ganando popularidad. No es para menos, ya que su sencillez, y al mismo tiempo su potencial, son parte fundamental de REST. HATEOAS fue más allá de las costumbres, siendo responsable de que la web se dejara de ver simplemente como almacenamiento y obtención de información, y pase a comportarse como una plataforma de aplicación, ejecutando tareas, y teniendo múltiples estados posibles. HATEOAS se diferencia de las aplicaciones distribuidas convencionales, donde uno puede ir realizando transiciones y avances, pasando de un estado a otro. Aquí el concepto es similar, sólo a que a diferencia de otras aplicaciones, los estados no se conocen de antemano. En REST, con la restricción que impone HATEOAS, cuando la aplicación alcanza un determinado estado, es éste el responsable de indicarle las posibles transiciones de estado a realizar. La hipermedia entra en juego ya que es la responsable de informar los estados a los que se puede transitar. Pensemos el siguiente ejemplo: Cuando estamos llenando un formulario en una web, y presionamos sobre el botón “Enviar” se produce una transición de estado, la cual fue “habilitada” al momento que entramos al sitio que contenía el formulario a llenar. Es muy posible que de antemano no supiéramos de la existencia de tal botón, ni del estado de la aplicación al que nos iba a transferir.

2.2 Modelo de maduración de Richardson

Leonard Richardson, ha diseñado un modelo el cual podemos describir en una palabra, como una “receta”. Se trata de unos pocos pasos o niveles a seguir, para que de manera sencilla e intuitiva, logremos cumplir las restricciones que propone REST.

A continuación se describen los niveles propuestos, los cuales se explican con un ejemplo sencillo: el escenario donde un paciente necesita conocer los turnos de los cuales dispone su doctor, para elegir alguno de entre los que están libres.

Nivel 0

Este nivel describe el punto de partida fundamental que se necesita para evolucionar hacia un sistema RESTFUL, es decir, un sistema que cumpla con todas las restricciones que propone REST. Aquí las exigencias son pocas. Es obligatorio usar HTTP como protocolo de transporte para las interacciones que realice el sistema. También se plantea la existencia de una única URI y de un sólo método HTTP, el cual usualmente es POST debido a su versatilidad. Bajo estas restricciones, nuestro sistema queda modelado más bien como RPC [12].

Haciendo uso de nuestro ejemplo, en el “nivel 0” el paciente (cliente del servicio) debería realizar una petición al sistema de turnos, junto con un cuerpo del mensaje en donde se brinde toda la información que se necesita: es decir, desde la identificación del doctor con el cual desea ser atendido, especialidad, horarios de preferencia. La respuesta a este llamado podría ser una lista que contenga los turnos disponibles para el profesional seleccionado. Como último paso, el cliente debería volver a llamar al servicio, esta vez con la información para confirmar uno de los turnos disponibles: doctor, información personal del paciente, y turno seleccionado. La respuesta a este llamado debe contener un cuerpo del mensaje que sirva para interpretar el éxito o no de la operación. Es decir que bajo este nivel, el cuerpo del mensaje tiene un rol decisivo para determinar la operación que se debe realizar.

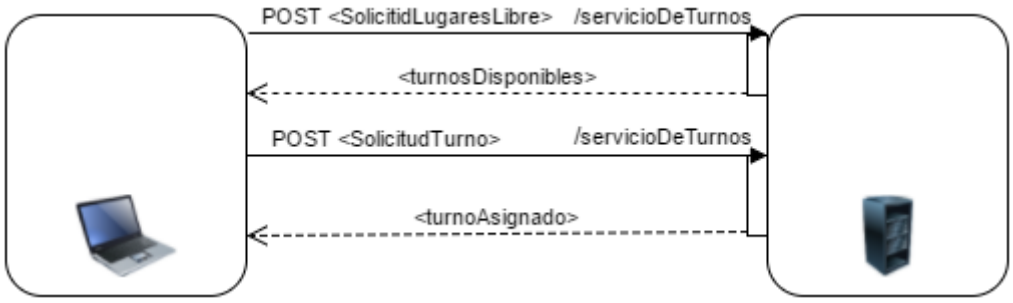


FIGURA 2.1 EJEMPLO DE INTERACCIÓN REST EN NIVEL 0

Sólo alcanza una simple observación en la figura 2.1 para detectar que el “nivel 0” está muy lejos de ser un sistema RESTFUL. Pero no deja de ser el primer paso hacia lo que Fowler denomina “la gloria de REST” [13]

Nivel 1

El siguiente nivel agrega la noción de “recursos”. De esta manera dejamos de tener una única URI para la identificación del sistema en un todo, para pasar a identificar a cada recurso bajo su URI única. En este sentido el sistema se vuelve más granular. Tengamos en cuenta que este es

el único cambio, por lo que seguiríamos usando un sólo método HTTP (en nuestro ejemplo, POST)

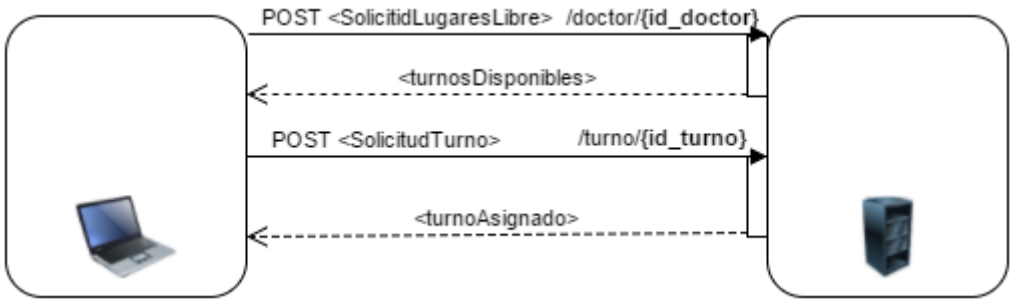


FIGURA 2.2 EJEMPLO DE INTERACCIÓN REST EN NIVEL 1

Siguiendo el ejemplo del doctor y los turnos, se notan algunas modificaciones, ilustradas en la figura 2.2: en el primer llamado debemos pedir el listado de turnos disponibles a un doctor en particular, identificado por un recurso único. Cuando conocemos el turno que nos interesa, debemos reservarlo enviando nuestra información personal al recurso de turnos, identificándolo de manera unívoca. La respuesta a este último llamado deberá contener un cuerpo de mensaje que nos permita conocer el éxito o fracaso de la petición.

Nivel 2

Aprovechando las virtudes de HTTP, el nivel 2 nos lleva a la correcta utilización de los diferentes "verbos" o métodos HTTP [14]. Cada método tiene su significado y su razón de ser en HTTP. Cumpliendo los requisitos del nivel 2, estaremos haciendo uso consciente de ese significado, llevando a nuestra aplicación un paso más cerca de ser RESTFul.

Por ejemplo, usamos GET para operaciones seguras, POST para crear nuevos recursos, etc. Los diferentes métodos disponibles, junto a significado serán explicados en detalle más adelante.

Otro punto que se introduce en el nivel 2, es el uso de los diferentes estados que brinda HTTP [15]. Con la utilización de los mismos, no sería necesario informar el estado de la interacción petición-respuesta, sino que podríamos usar alguno de los códigos HTTP conocidos. Por ejemplo, cuando un GET es exitoso, deberíamos devolver un estado *HTTP 200 OK*. Cuando el servidor falla por algún motivo, deberíamos devolver un estado HTTP *500 Internal Server Error*. Luego, en el cuerpo del mensaje se podría detallar el error, indicando las causas.

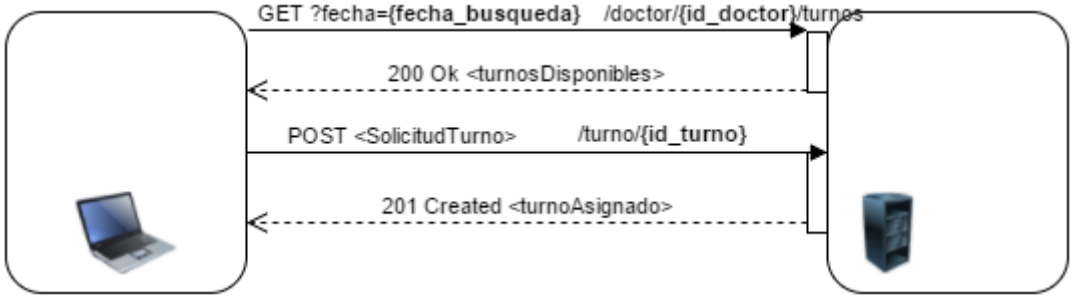


FIGURA 2.3 EJEMPLO DE INTERACCIÓN REST EN NIVEL 2

En nuestro ejemplo, bajo la primera llamada desaparece el cuerpo del mensaje, ya que los datos los podemos pasar como parámetros en la URL utilizando un método HTTP GET, dado que deseamos obtener información de un recurso, en este caso, los turnos disponibles para el doctor

que identifiquemos. Si todo ocurre con normalidad, recibiremos una respuesta con un estado HTTP 200, cuyo cuerpo del mensaje nos brindará la información de turnos en base a nuestros filtros. El último paso no varía tanto, ya que la petición debe incluir también la identificación del doctor y nuestros datos personales, para poder reservar el turno efectivamente. Lo que sí varía, es la respuesta a este servicio, que introduce varios estados HTTP para identificar el éxito o no de la operación. Opcionalmente se podría incluir un detalle de la operación, como por ejemplo el identificador del turno para luego poder consultarlo, o el código de error en caso de que existiera alguno. Esta situación se ve ilustrada en la figura 2.3

Nivel 3

El último nivel nos abre paso al ya mencionado concepto de HATEOAS. Se busca que cada recurso nos indique, con la utilización de hipermedia, qué es lo próximo que podemos hacer, hacia dónde ir, o cuáles son las transiciones de estado que permite nuestra aplicación.

En el caso ideal, el usuario sólo debería conocer la URI que sea punto de partida, y con la utilización de HATEOAS debería ser capaz que navegar, e ir conociendo los estados de la aplicación a medida que avanza. Esto posee claras ventajas, como por ejemplo la posibilidad de que las URIs vayan mutando y evolucionando sin tener impacto mayor en el usuario. También hay algunos puntos a tener en cuenta, como la necesidad de que el usuario sepa interpretar la respuesta hipermedia. Por eso la importancia de tener ciertas restricciones.

Vale la pena mencionar que no hay ningún estándar en cómo representar el concepto de Hipermedia, aunque una opción bastante aceptada por la comunidad es la que presenta ATOM [16]

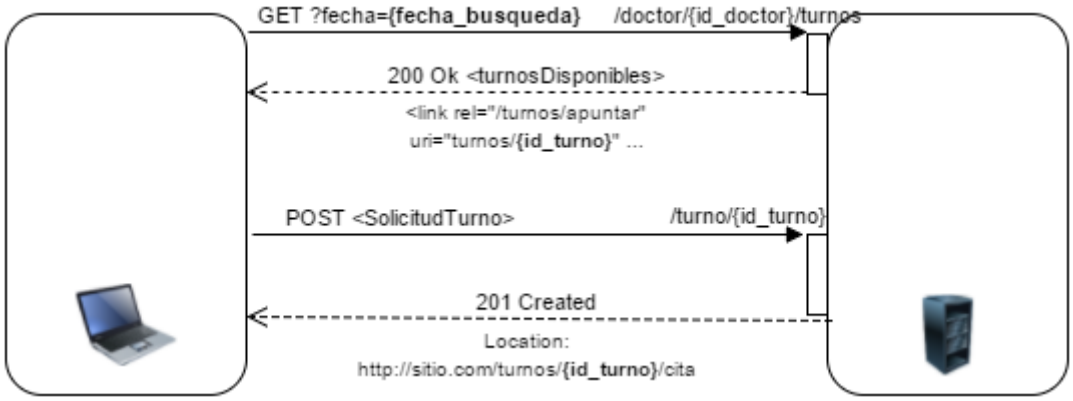


FIGURA 2.4 EJEMPLO DE INTERACCIÓN REST EN NIVEL 3

En el ejemplo del doctor y sus turnos, se sigue lo mencionado en el nivel 2, con algunas excepciones que se muestran en la figura 2.4: El usuario ahora navegaría hacia el turno que le interesa a partir de la respuesta otorgada en el servicio de turnos, conociendo cómo acceder a cada uno. Además, cuando se crea exitosamente el turno existiría una manera más clara de que el usuario acceda al turno nuevo, a partir de un encabezado de la respuesta HTTP.

2.3 HTTP - Métodos y códigos de respuesta

En pos de cumplir y aplicar correctamente las restricciones propuestas por REST, se aprovecha del protocolo de aplicación HTTP (Hypertext Transfer Protocol) sobre el cual se implementa. En

este sentido resulta fundamental comprender el significado de dos aspectos de HTTP: Sus métodos (verbos), y sus códigos de respuesta (estados)

Métodos

Métodos seguros

Se dice que un método es seguro cuando no es esperado que produzca efectos secundarios. Estos métodos deben ser utilizados sólo para obtener información, por lo que no deben producir cambios de estado en el servidor.

Métodos idempotentes

Un método es idempotente cuando el resultado de múltiples peticiones es el mismo que al ejecutar una petición simple. Esto se refiere al resultado de una petición-respuesta completa. Es decir, en el medio puede haber diferencias en el modo de procesamiento, pero el resultado final será el mismo.

A continuación se describen los métodos y se los categoriza según su definición. Algo a tener en cuenta, es que si bien un método se caracteriza por seguro, no-seguro, idempotente o no-idempotente, esta situación no se fuerza ni por el protocolo ni por el servidor. Por lo tanto podría pasar que un método "seguro" produzca cambios de estados en el servidor si su implementación no es la adecuada.

- OPTIONS: El método OPTIONS se utiliza para solicitar información al servidor. Particularmente se ejecuta sobre una URI para saber las operaciones que se pueden realizar en la misma. Por ejemplo podríamos conocer los tipos soportados, las operaciones, cabeceras soportadas, etc. El siguiente ejemplo nos muestra los encabezados que nos brinda. Entre ellos se encuentra "Allow", el cual indica los métodos soportados para la URI en cuestión

```
Server: Apache/2.4.1 (Unix) OpenSSL/1.0.0g
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Type: httpd/unix-directory
```

- GET: Este método tiene uno de los propósitos más claros, el cual es el de obtener la representación de un recurso identificado en la URI. Debería ser un método seguro e idempotente en su óptima implementación. Existen algunas variantes al GET convencional
    - GET Condicional: Un GET-Condicional devolverá un resultado exitoso sólo en caso de cumplir con alguna de las condiciones que se le pueden proveer en algunas cabeceras (headers) reservadas. Las cabeceras más populares son: *If-Modified-Since, If-Unmodified-Since, If-Match*, entre otras.
    - GET Parcial: El GET-Parcial transportará en su respuesta sólo parte del recurso solicitado.

  Tanto el GET-Condicional como también el GET-Parcial tienen como objetivo, no sólo mejorar la carga en el servidor, sino también aliviar el tráfico innecesario en la red.

  Tener en cuenta que las respuestas de las peticiones GET pueden ser cacheadas sólo si se indica esto mediante una cabecera especial.

- HEAD: Considerado un método seguro e idempotente, HEAD se comporta de manera idéntica a GET, sólo que no devuelve ningún cuerpo de mensaje. Es decir, de un HEAD sólo se obtendrían cabeceras, las cuales deben ser idénticas a la petición hecha por el método GET a la misma URI. Este método, al no retornar un cuerpo de mensaje, es utilizado frecuentemente para consultar cierta meta-información de un recurso, como por ejemplo, la fecha de última modificación.
- POST: Junto con GET, uno de los métodos más populares y aceptados en la Web. Si bien la verdadera función del método la determina el servidor, su propósito general es que el servidor origen acepte ciertos datos enviados en el cuerpo (una entidad), identificando al recurso mediante la URI de petición. De esta manera los usos son varios, desde enviar un nuevo mensaje a un foro de debates, un nuevo mail a una lista de difusión, o lo más popular al momento, enviar los datos de un formulario servido en un sitio HTML [17].
- PUT: En algunos aspectos similar a un POST, el método PUT solicita que la entidad que se envía en el cuerpo del mensaje se almacene bajo el recurso identificad en la URI. Ahora bien, si la entidad ya existía, se puede entender como una actualización de la misma. Si la entidad no existía, y puede ser creada, debería hacerse e informarse. Del mismo modo, cuando la entidad no existía, y no puede ser creada, el servidor debería informar esta situación. Todos estos escenarios pueden ser contemplados y diferenciados haciendo uso de los diferentes códigos HTTP.
- DELETE: Una petición realizada con el método DELETE solicita que se elimine un recurso identificado bajo la URI de petición. Si bien el comportamiento de este método puede ser reemplazado por otro (por intervención humana), el mismo no debería responder un estado exitoso a menos que no haya realizado la acción de eliminar el recurso solicitado (o al menos lo haya movido a una locación inaccesible). Opcionalmente se puede devolver el cuerpo de la entidad afectada. Este método es considerado no-seguro.
- TRACE: Considerado como un método para debug o testing, TRACE se comporta de manera muy peculiar. Lo que hace es imprimir en el cuerpo del mensaje de respuesta, la petición original. Es decir, devuelve al cliente, en el cuerpo, la petición original, tal cual fue generada. De esta manera se podría detectar, por ejemplo, si existe algún paso intermedio (proxy) que esté modificando la petición.
- CONNECT: Este método se reserva para usos de tunneling. Sobre todo para proveer conexiones seguras, HTTPS [18] , cuando existe, en la ruta, un proxy que no es seguro.

Códigos de respuesta

La primera línea en una respuesta HTTP es el *código de respuesta* [19]. Este código puede venir acompañado de una pequeña descripción, la cual a veces se utiliza, en el caso de error, para indicar la causa. Sin embargo no es recomendado, ya que el estándar indica lo siguiente:

- El código de respuesta puede ser leído e interpretado por una máquina
- La descripción del código de respuesta puede ser leído e interpretado por un humano.

La importancia del código de respuesta radica en cómo el cliente puede y/o debe manejar esa respuesta. No sólo influye el código de respuesta, sino que también otras cabeceras pueden modificar el procesamiento.

La definición permite utilizar otros códigos de respuesta que no sean los incluidos en el estándar. En este caso, si el cliente no sabe a qué se refiere el código de respuesta, utilizará su primer dígito para entender a cuál "familia de mensajes" se refiere:

- 1xx – Informational: Esta familia de códigos indican respuestas *provisionales.* En estas respuestas sólo viene el código, y pueden venir cabeceras opcionales. Los clientes deben estar preparados para recibir una o más de estas respuestas antes de obtener la respuesta definitiva. El ejemplo más popular de estos códigos es el *101 - Switching protocols*, el cual indica que la petición debe continuar, pero bajo la actualización de protocolos, mediante la cabecera "Upgrade"
- 2xx – Successful: Esta familia de códigos de respuesta es la más popular. Se utiliza para indicar que la petición fue recibido o aceptado satisfactoriamente. Se diferencian en *200 OK, 201 Aceptado, 204 Sin Contenido, 206 Contenido Parcial*, entre otros.
- 3xx – Redirection: Bajo una respuesta que contenga un código de la familia "3xx", el cliente necesitará tomar una acción para completar la petición. Esta acción tal vez puede ser llevada a cabo de manera "automática" o transparente al usuario final, a través del user-agent (por ejemplo el navegador web), aunque en algunos casos se necesita de intervención manual. Los usos más populares de estos códigos se dan en casos donde un recurso "cambió de lugar" a una nueva URI; en este ejemplo, mediante un código de respuesta *301 - Movido permanentemente*, junto con la cabecera *Location*, el cliente podrá completar la acción, conociendo la nueva ubicación del recurso. También puede ocurrir que un recurso haya sido temporalmente re-ubicado, en cuyo caso se debería responder con un código *307 - Redirección temporaria*. Otro código de la familia 3xx que resulta muy popular es el *304 No Modificado*. El código se utilizará para saber cuándo un recurso fue modificado o no, y de esta manera se puede optimizar el tráfico de red, utilizando algún mecanismo de caché.
- 4xx - Client Error: Cualquier código de la familia 4xx indica que el usuario cometió algún error el enviar la petición. El servidor debe enviar en estos casos, un cuerpo de mensaje que especifique cuál fue el error del usuario (esto se da siempre, excepto cuando se utiliza un método HEAD). Ejemplos de esta familia de códigos son: *400 - Mala petición*, el cual indica que la petición no puede ser entendida por el server por un error de sintaxis; *401 Sin-Autorización*, el cual indica que para completar la petición se requiere autorización. En este caso se debe incluir la cabecera WWW-Authenticate. El más conocido de esta familia es el famoso *404 - No encontrado*, que significa que el recurso identificado bajo la URI de petición no existe
- 5xx - Server Error: En este grupo de errores, identificado con el primer dígito 5, corresponde a situaciones en las que el servidor sabe que tiene un error o que es incapaz de efectuar la petición. En estos casos también se debe adjuntar un cuerpo de mensaje que detalle el error, con excepción de las peticiones realizadas con el método HEAD. Entre los ejemplos más populares se deben mencionar: *500 - Error interno del servidor*: En este caso el servidor se encontró con una condición inesperada, la cual imposibilitó que se finalice correctamente con la petición; *501 - No implementado*: Este código

resulta conveniente que sea utilizado cuando el servidor no soporta la funcionalidad requerida y se le imposibilita completar la petición, aunque también es muy útil para cuando el servidor no reconoce el método de la petición o no lo soporta para ningún recurso.

Se han nombrado y desarrollado en este capítulo las principales características ofrecidas por REST. Los ejemplos mencionados acompañan cada descripción para plasmar de manera clara el uso real que se le da a cada concepto.

Es fundamental para cualquier desarrollador de software comprender y aplicar con criterio todos estos conceptos, pues hoy un sistema no funciona si no es con interacción con muchos otros sistemas desarrollados por terceros. En ese sentido, REST nos brindará la interfaz (web) para entendernos con los diferentes sistemas con quienes necesitemos interactuar.

Luego, en el desarrollo de las aplicaciones que acompañan a esta tesina, se verán plasmadas la mayoría de las recomendaciones aquí descriptas.

Capítulo 3 - CLOUD COMPUTING

Los cambios de paradigmas que se vienen produciendo en los últimos años en el sector de las ciencias de la computación, llevaron a la definición de nuevos términos asociados no solo con la infraestructura de hardware, sino también con el software, los servicios, y la combinación de ellos. Así surgieron nuevas tecnologías, nuevos paradigmas, y diferentes visiones que ya no solo involucran a los profesionales informáticos, sino que alcanzan a una visión global, donde toda una organización suele tener injerencia.

En el presente capítulo se definirá el término “cloud computing” o “computación en la nube”, no sin hacer antes un pequeño recorrido por sus antecedentes. Luego se hará un recorrido brindando las principales características de este nuevo término.

Por último, se categorizarán los diferentes tipos de nubes, bajo dos criterios; en primer lugar se caracteriza a la nube de acuerdo a sus diferentes modelos de despliegue. Luego, se dará una explicación sobre los diferentes (y más populares) modelos de servicio que ofrece la computación en la nube.

3.1 Definición

Sucede en varias ocasiones dentro de las ciencias de la computación, que la tarea de definir un término no resulta muy clara. Es así el caso del término “cloud computing”. Esto se puede explicar por varios motivos. El principal, es la evolución rápida y constante de la computación en todo su aspecto. Pero además debemos sumarle que muchos de los términos que manejamos son producto de la combinación de varios otros términos anteriores, que fueron, nuevamente, evolucionando y dando vida a nuevas definiciones.

El término “cloud computing” bien se puede definir nombrando previamente a sus predecesores de manera breve:

Grid Computing: Se trata de un paradigma de cómputo distribuido en el cual se coordinan recursos conectados mediante una red para lograr un objetivo en común. Originalmente surge para uso científico, en aplicaciones que requieren cómputo intensivo. De la definición de Grid Computing tomaremos el concepto que establece que *se utilizan recursos distribuidos de hardware para alcanzar objetivos a nivel de aplicación*.

Utility Computing: Se refiere a una forma de proveer recursos de hardware bajo demanda, cobrando al usuario sólo por ese uso, en lugar de que se le provea de una tarifa única. Utility Computing se considera el paso previo inmediato al nacimiento del Cloud Computing tal como lo conocemos hoy.

Virtualización: El término “virtualización” nos ofrece una abstracción inmediata del hardware verdadero que tenemos disponible. Se dice que la virtualización ofrece recursos virtuales listos

para ser utilizados por la aplicación de alto nivel. A un servidor que se conformó por recursos virtualizados se lo denomina "*máquina virtual*", o más popular su término en inglés, "*virtual machine*" o simplemente "*vm*" Esta definición resulta fundamental a lo que hace cloud computing, ya que habilita al hecho de lograr, de forma dinámica, obtener y desechar recursos bajo demanda.

Automatic Computing: Definido así por IBM en el año 2001 [20], el término apunta a obtener sistemas capaces de auto administrarse. Un ejemplo claro es un sistema que reacciona ante ciertos eventos (por ejemplo, exceso uso de CPU, poca memoria RAM disponible, pérdida de conexión de red, etc.). Cloud computing toma esta definición, que originalmente fue establecida para aliviar la carga de la administración de sistemas complejos, y la aplica para reducir la utilización innecesaria de recursos, reduciendo directamente los costos.

Ya conociendo bajo qué conceptos se origina el término *cloud computing*, resulta conveniente citar la definición que brinda The National Institute of Standards and Technology [21]:

> *La computación en la nube (cloud computing) es un modelo que permite el acceso conveniente, a través de la red, a un conjunto compartido de recursos configurables (redes, servidores, almacenamiento, aplicaciones, y servicios, entre otros). Estos recursos pueden ser provisionados rápidamente, y liberados con un mínimo esfuerzo de administración y casi sin interacción con el proveedor de servicios.*

De esta manera logramos obtener una definición simple, pero a la vez certera sobre el término cloud computing. Como se puede apreciar, la definición no agrega ninguna tecnología que haya sido inexistente hasta el momento.

Acordar sobre una definición no es fácil, ya que nos podemos encontrar con más de 20 variantes [22] que intentan definir a la computación en la nube.

3.2 Principales características

Pese a la variedad de definiciones con las que podemos encontrarnos para el término "cloud computing", entre todas encontramos características comunes, las cuales se enumeran a continuación:

Servicio bajo demanda: Es el cliente quien decide, de manera independiente y sin requerir intervención humana, cuando necesita aprovisionarse de nuevos recursos.

Alta disponibilidad de acceso a la red: El acceso a la red es amplio, ya que se accede a través de mecanismos estándares. Esto permite que se acceda por ejemplo con APIs que trabajan sobre HTTP permitiendo así el acceso a un gran rango de dispositivos (PCs, smartphones, etc)

Agrupación de recursos: Todos los recursos de cómputo que ofrece el proveedor, son agrupados para servir a múltiples consumidores bajo el modelo de "tenencia múltiple" [23]. De esta manera, los recursos se asignan y reasignan de acuerdo a la demanda de los consumidores.

Capacidad de rápida elasticidad: La computación en la nube ofrece la característica de auto controlar la cantidad de recursos que necesita. De esta manera se auto-provisiona cuando necesita algún recurso en particular (por ejemplo, adquiriendo más núcleos de procesador ya que la carga media supera el 80%), como así también libera recursos cuando están ociosos o no

los necesita. Esta característica brinda al consumidor la ilusión de que posee una cantidad ilimitada de recursos. También facilita la estimación inicial de costos, y logra que no se desperdicien recursos, lo cual es muy valioso ya que eso se traduce en ahorro directo de costos. Otro punto a favor de esta característica es que se desarrolla mientras mantienen altos niveles de confiabilidad y seguridad de toda la infraestructura.

Servicio mensurable: Los proveedores de servicios de computación en la nube usan sistemas de métricas que permiten cobrar al usuario solamente por lo que ha usado en un período determinado de facturación. Se le brinda un informe detallado con los recursos, su costo, y el tiempo de utilización, para que de esta manera el consumidor tenga en claro en qué se basa su costo total.

3.3 Modelos de despliegue

La computación en la nube posee varios criterios para ser categorizada. Uno de ellos es su "modelo de despliegue" [24], que va a categorizar exactamente al entorno en donde se desarrolla la nube y la va a distinguir de acuerdo a la propiedad de los datos y los servicios, el tamaño, y el acceso que se provea. Esta categorización también suele orientar sobre la naturaleza de la nube, su fin, y su potencial uso.

Nube Privada

También conocida como "nube interna". En este modelo la nube se despliega, mantiene y se opera para una organización específica. El propietario, quien la administra, y quien la opera pueden, sin embargo, ser de la misma organización o de una tercera organización involucrada específicamente para este fin. Bajo esta infraestructura, la red se encuentra protegida mediante un firewall el cual es administrado por el correspondiente departamento de IT. En este tipo de nubes se restringe el acceso sólo a usuarios autorizados. La organización gana así mayor potestad sobre sus propios datos, evitando el acceso por parte de usuarios indeseados que puedan perjudicar a la organización haciendo un mal uso de los mismos.

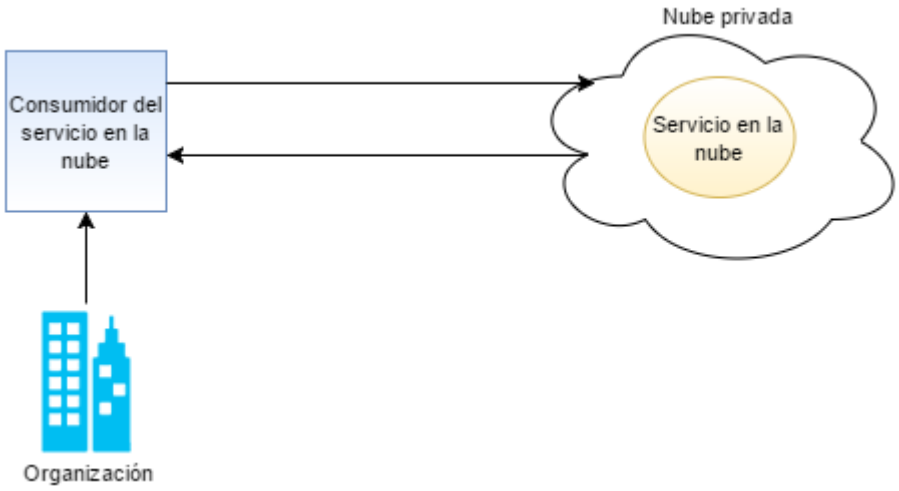


FIGURA 3.1 ESQUEMA DE NUBE PRIVADA

Nube Comunitaria

En este modelo, la infraestructura necesaria para generar la nube se provisiona exclusivamente para ser usada por una comunidad de consumidores específica que provienen de diferentes organizaciones con un mismo fin. Este tipo de nubes pueden ser administradas y operadas por uno o más integrantes de las diferentes organizaciones de la comunidad, por un tercero, e incluso por una combinación de estos dos. En estos casos la infraestructura es compartida entre la comunidad interesada, compartiendo también los costos, lo que produce un beneficio económico. Este modelo de implementación en la nube es ideal para situaciones en las cuales existen más de una organización, las cuales comparten un fin en común o deben implementar proyectos similares.

FIGURA 3.2 ESQUEMA DE NUBE COMUNITARIA

Nube pública

En el modelo de servicio de nube pública, la infraestructura está disponible mediante la red al uso del público general, en algunos casos mediante un acuerdo comercial. Esto habilita al consumidor/cliente a crear servicios y desplegarlos en la nube con un costo mínimo comparado con los costos que ofrecen otros tipos de infraestructura. Muchas veces se afirma que este modelo de implementación representa a la nube en su más puro estado por su modo de comunicarse y proveer los servicios de manera masiva. Desde el punto de vista técnico, casi no existe diferencia con el modelo de nube privada; los rasgos más distintivos se dan a nivel de seguridad y autenticación necesaria para acceder y consumir los diferentes recursos disponibles.

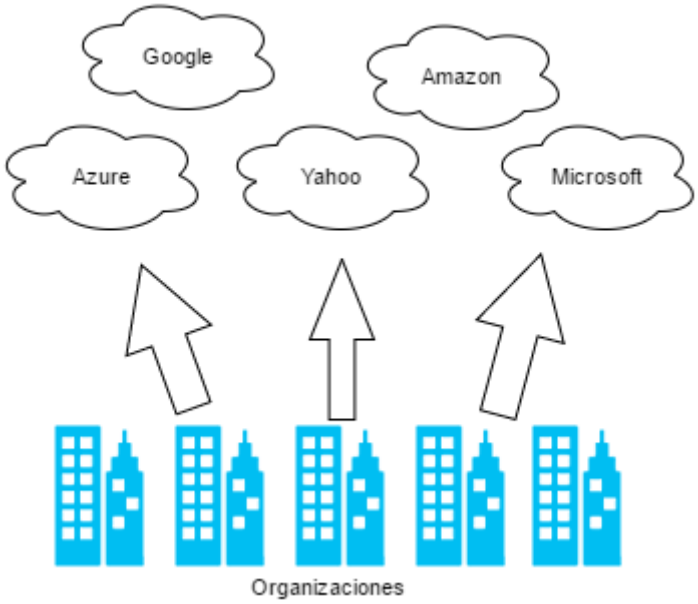


FIGURA 3.3 ESQUEMA DE NUBE PÚBLICA

Nube híbrida

Como su nombre lo indica, este modelo de computación en la nube se forma por la combinación de dos o más infraestructuras formadas por los modelos ya mencionados (nube privada, nube pública, nube comunitaria). Cada infraestructura puede tener la capacidad, a través de sus interfaces de red, de transferir datos y aplicaciones desde una infraestructura hacia otra.

Suele ser muy útil en casos donde, por ejemplo, una organización necesita proveer servicios públicos, pero decide el almacenamiento de sus registros en una infraestructura privada para proteger los datos de sus clientes.

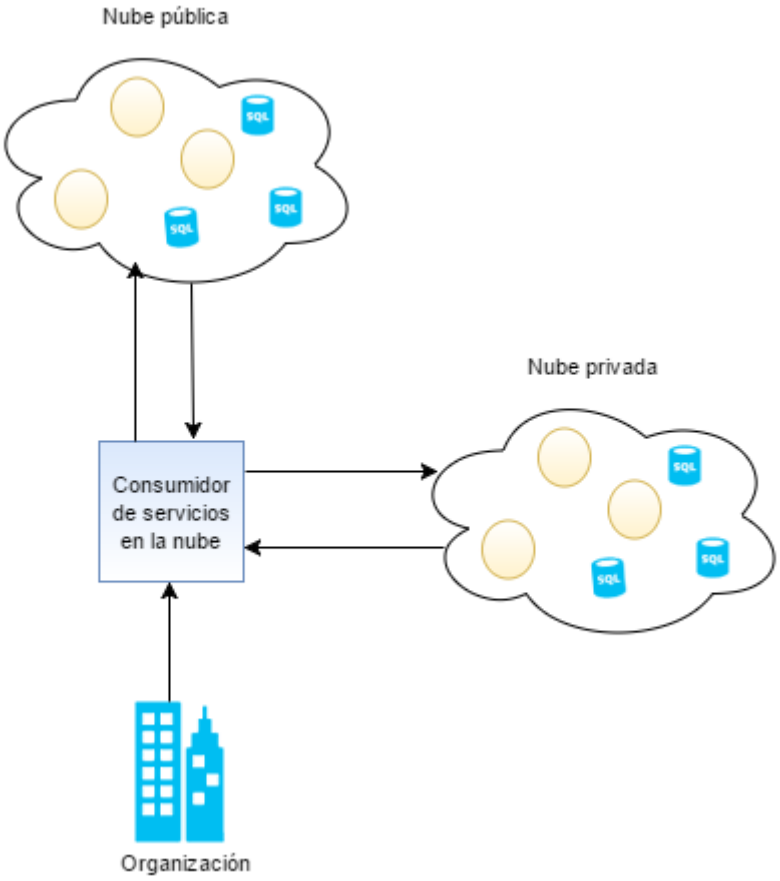


FIGURA 3.4 ESQUEMA DE NUBE HÍBRIDA

3.4 Arquitectura en la nube y modelos de servicio

En la computación en la nube existen diferentes modelos de servicio, los cuales componen una visión sobre su arquitectura. Estos modelos proponen diferentes niveles de abstracción, cada uno con sus características. Todos ellos se caracterizan dentro de una arquitectura orientada a servicios [25]

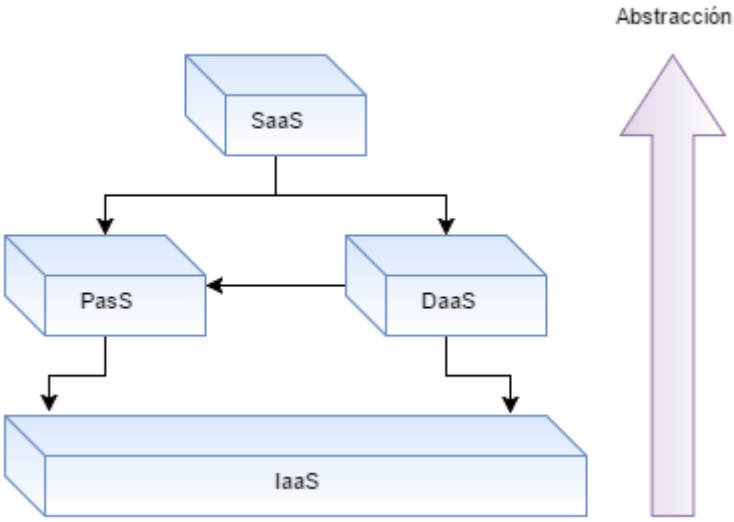


FIGURA 3.5 MODELOS DE SERVICIO EN LA NUBE

Software como Servicio (SaaS - Software as a Service)

También conocido como "servicio bajo demanda". Generalmente SaaS es accedido por los usuarios usando un cliente liviano, por ejemplo a través de un navegador web. Es el modelo más usado y popular de la computación en la nube. Se ubica en la capa más alta de abstracción. Existe un acuerdo contractual, mediante el cual el vendedor (proveedor de servicio) es totalmente responsable de proveer la infraestructura esencial, la cual consiste de recursos robustos de hardware y de sistemas de software escalables. Un uso muy popular que se suele dar a SaaS, es también por parte de los desarrolladores, quienes encuentran la posibilidad de integrar servicios de la nube a sus propios desarrollos.

Se dice que es el modelo de más abstracción ya que aquí, los proveedores de la nube son quienes administran no sólo la infraestructura, sino también las aplicaciones que proveen. El usuario simplemente accede a ellas a través de un único punto de acceso, desconociendo así la infraestructura subyacente.

Como ventaja inmediata a este modelo, las empresas obtienen el potencial de reducir el costo operacional del departamento de IT, terciarizando el mantenimiento y soporte del hardware y el software a un proveedor de computación en la nube.

Datos como Servicio (DaaS - Data as a Service)

DaaS [26] es un tipo de servicio de computación en la nube que provee datos bajo demanda, generalmente expuestos para que sean consumidos por los clientes a través de APIs, ya sea para descargar archivos o hacer diferentes consultas de datos. Utilizando este modelo de servicio, los clientes no necesitan obtener y almacenar conjuntos gigantescos de datos.

Con este modelo las organizaciones se liberan del alto costo que pueden tener los motores de base de datos y el almacenamiento en masa.

Plataforma como Servicio (PaaS - Platform as a Service)

En el modelo de plataforma como servicio, el proveedor del servicio es responsable de proveer un entorno robusto para el desarrollo de software. Existen varias PaaS que habilitan luego el acceso a recursos vía IaaS. PaaS generalmente ofrece los denominados “toolkits” de desarrollo, con lo que no sólo es posible desarrollar, sino también realizar la puesta en producción de una aplicación.

Los desarrolladores de aplicaciones pueden desarrollar y ejecutar sus soluciones de software en una plataforma de computación en la nube sin el costo y la complejidad de comprar y administrar las capas de software y hardware subyacentes.

Ejemplos de este modelo de servicio son Google App Engine o Microsoft Azure

Infraestructura como Servicio (IaaS - Infrastructure as a Service)

Este modelo de servicio se refiere a la capa más baja de abstracción. Se lo considera el modelo de servicio “más básico” de la nube. IaaS ofrece a sus consumidores servicios de infraestructura, es decir, recursos de hardware tales como CPU, memoria, almacenamiento, redes, balanceadores de carga, entre otros. Los ofrece de manera física, pero también de manera virtual, en forma de máquinas virtuales. Sin este modelo no se es posible darle vida a las abstracciones ya mencionadas (PaaS, DaaS, SaaS). Algunos ejemplos populares que implementan IaaS son Amazon Web Services, Rackspace.

Los proveedores de IaaS ofrecen sus recursos bajo demanda, obteniéndose de uno de sus data-centers, los cuales se componen de grandes pools de equipamiento. El costo de este modelo de servicio se da generalmente por la cantidad de recursos alocados y consumidos.

El usuario de este modelo de servicio tendrá para elegir diferentes variantes de los recursos de hardware y software; será responsable de mantener los sistemas operativos actualizados y configurados para ejecutar sus aplicaciones.

3.5 Alternativas comerciales

Se presentan y comparan aquí, dos de las más populares alternativas comerciales en lo que se refiere a computación en la nube. Una de ellas es *Amazon Web Services*, la cual, pertenece a la compañía Amazon. La segunda alternativa que se presenta es *Microsoft Azure*, perteneciente al gigante Microsoft.

La comparación de proveedores en la nube no sólo la debemos realizar en cuestión de costos (el cual es un factor muy importante, y en muchas organizaciones, el determinante), sino también en la calidad y alternativas de los diferentes servicios ofrecidos: desde variedad de hardware para elegir, los sistemas operativos pre-configurados (imágenes), opciones de red, formas de desplegar una aplicación, entre otros.

La compañía Gartner [27], líder en investigación y consultoría en tecnologías de la información, realiza todos los años un estudio exhaustivo de las empresas más influyentes en cuanto a computación en la nube. En este informe se incluye un típico gráfico cuadrante que la compañía desarrolla, en el cual se destacan 4 características, y se ubican a las diferentes empresas

conforme cumplan cada una de esas características: ‘desafiante’, ‘buen jugador’, ‘visionario’ y ‘líder’.

En lo que se refiere a la computación en la nube, Gartner realiza un análisis desde el punto de vista de la Infraestructura como servicio (IaaS), ya que este punto es el más abarcativo hacia el resto de las características “-como servicio”.

FIGURA 3.6 CUADRANTES DE GARTNER

Claramente los líderes de mercado aquí son Amazon Web Services y Microsoft Azure, como se ilustra en la figura 3.6. Este último ha ido ganando terreno en los últimos años, pues durante años el único líder del mercado ha sido Amazon. Sin embargo durante el último tiempo esto fue cambiando, en parte porque Microsoft ha hecho foco en avanzar sobre algunas desventajas que tenían los servicios de Amazon. Principalmente, la dificultad de navegación que presentaban algunos servicios y las interfaces no muy amigables. En este sentido, Microsoft ha sacado amplia ventaja con su interfaz gráfica mucho más amigable que la ofrecida por Amazon.

Luego, las compañías ofrecen productos similares: ambas dan soluciones bajo Windows o Linux. Cada una ofrece su plataforma de desarrollo: Azure ofrece la suite de Visual Studio y un producto denominado Azure SDK, el cual es un potente IDE (integrated development environment); Amazon ofrece CodeCommit para el control de versiones, CodeDeploy para la automatización

del despliegue de aplicaciones y una interfaz de línea de comando con diversas opciones para interactuar con la mayoría de las componentes de Amazon.

Tanto Amazon con Azure ofrecen soluciones virtualizadas con una gran variedad de alternativas en cuando a núcleos, memoria RAM, disco. Para el almacenamiento a gran escala Amazon tiene su popular servicio “Amazon’s simple storage service (s3)”, mientras que su par en Azure tiene una variedad de opciones, la más popular es “blob storage”.

Además, ambas compañías brindan también “almacenamiento histórico” o “frío”. En el caso de Azure se llama Azure Backup, mientras que en Amazon se denomina Amazon Glacier. Ambos servicios tiene como fin el almacenamiento histórico, en frío, de datos que no son frecuentemente accedidos.

En lo que se refiere al costo del servicio, ambas compañías ofrecen períodos gratuitos de prueba. Esto es muy útil a la hora de tomar una decisión de acuerdo a nuestras necesidades. Por las similitudes que pueden llegar a tener algunos servicios, la elección no es tarea fácil: haciendo uso de los servicios gratuitos, se tendrá un panorama general de la calidad de servicio ofrecida por cada opción.

Sin embargo, una vez terminado el período gratuito, el uso por cada servicio comienza a ser pago. En la mayoría de los casos el costo es un factor decisivo para elegir una compañía u otra, y por esto ambas ofrecen un servicio de “calculadora” para tener una aproximación del costo que puede generar montar nuestro sistema.

La calculadora ofrecida por Microsoft Azure [28] tiene una interfaz muy amigable y es intuitiva al usuario. En cambio la ofrecida por Amazon Web Services [29] no tiene la misma suerte. Su interfaz resulta avanzada y con muchas opciones para el usuario promedio. De todas maneras se logra su uso, aunque se recomienda tener un conocimiento previo de los servicios que ofrece Amazon.

Capítulo 4 - BIG DATA

Google [30], Facebook [31], Amazon [32], Linkedin [33], Netflix [34], Spotify [35] Estos sitios, entre tantos otros, considerados naturalmente sitios comerciales o redes sociales, tienen algo en común. Aunque mayoritariamente el uso que se les dé sea de entretenimiento u ocio, existe un factor clave que tienen en común, el cual es un pilar clave para su existencia y para que sean atractivos no solo a los usuarios, sino, principalmente a los inversores que los mantienen con vida. Pues esta clase de sitios utilizan una serie de algoritmos y análisis de datos que escapan al análisis tradicional de hace una década atrás. Estos nuevos paradigmas y técnicas permiten el análisis y la interpretación de inmensas cantidades de datos, tarea que sería imposible de realizar con la utilización de los procesos y herramientas convencionales, y que al mismo tiempo son fundamentales para su funcionamiento.

En el presente capítulo se llevará a cabo la tarea de definir “big data”, tarea que no es fácil ya que el término no posee una definición formal o académica. Muchas compañías y entidades académicas han dado sus definiciones. Se unificarán los principales aportes, extrayendo los puntos en común más relevantes.

Con una definición clara, se enumeran las principales características que hacen a “big data”, haciendo hincapié tanto en las características que debe poseer el software, el hardware, y también nombrando particularidades que deben tener los datos.

Seguiremos luego con la presentación de Map Reduce, un modelo de programación clave para entender e implementar un eficiente sistema que sea considerado “Big Data”.

Para finalizar se ilustran estos conceptos con ejemplos claros y sencillos.

4.1 Definición

El importante avance de la tecnología ha obligado a las diferentes organizaciones, lucrativas o no, a modificar sus metodologías de análisis y explotación de datos. La primera pregunta que surge de inmediato ante esta afirmación es: ¿por qué sucede esto?

Pensemos por un minuto en la inmensa cantidad de información que produce una persona en el transcurso de un día de rutina, o en nosotros mismos: nos levantamos, escuchamos nuestra música favorita a través de algún servicio del tipo Spotify, obtenemos el estado del clima de acuerdo a nuestra ubicación en el GPS, leemos las noticias de nuestro diario favorito, y como si fuera poco, accedemos a redes sociales como Facebook donde indicamos nuestros gustos, preferencias y costumbres acerca de la comida, literatura, música, etc. Aprobamos o desaprobamos artículos, opiniones, ideologías. Luego durante el transcurso del día realizamos alguna compra o transacción comercial a través de la banca electrónica.

Sólo este ejemplo, ha dado una vasta información a los diferentes involucrados para que “nos conozcan”. Pero ¿qué beneficio nos daría esto? Aquí tenemos que considerar dos puntos de vista: si consideramos el punto de vista del usuario, se logra una experiencia mucho más

personalizada, agradable, apuntada directamente a nuestra necesidad. Sólo obtenemos lo que queremos; visto desde el punto de vista lucrativo o comercial, se apuntaría a mostrar al usuario lo más adecuado para él, con mejores chances de lograr un incremento de ingresos por ventas o publicidad personalizada.

Pero los ejemplos no deben limitarse al área comercial: pensemos en cuánto tiempo se podría ganar en el diagnóstico de enfermedades al procesar grandes volúmenes de datos de manera eficiente; o cómo se podrían planificar los accidentes naturales y sus evacuaciones, como es el caso de los terremotos o tsunamis.

Definir el término big-data no es tarea fácil, principalmente por el uso bastante amplio que se le ha dado en el último tiempo. La mayoría lo asocia directamente a "grandes cantidades de información". Algunos simplemente lo definen como un modelo de negocio o como información en tiempo real.

IBM realizó una encuesta [36] a un grupo de personas de diferentes profesiones, donde se les solicitaba elegir dos características a las que ellos consideren como definición de big-data. El resultado de la encuesta fue bastante curioso, pues ninguna característica predomina sobre el resto:

- Mejor acceso a la información (18%)
- Nuevos tipos de datos y análisis (16%)
- Información en tiempo real (15%)
- Influencia de datos generados por nuevas tecnologías (13%)
- Medios formados de manera no tradicional (13%)
- Grandes volúmenes de datos (10%)
- El último término de moda (buzzword) (8%)
- Datos originados en medios sociales (7%)

Jonathan Stuart y Adam Barker, en su artículo [37] definen a Big Data como,

> […] el término que describe el almacenamiento y análisis de grandes y complejos conjuntos de datos, usando una serie de técnicas que incluyen, (aunque existen otras), NoSql, MapReduce y machine-learning

La definición se forma luego de analizar las interpretaciones realizadas por grandes figuras en el campo de la investigación informática, como son IBM, Microsoft, Google y Oracle [38]

Concluyen en la citada definición luego de comprender que todas las definiciones compartían los siguientes puntos:

- Tamaño: El volumen de los conjuntos de datos es un factor crítico
- Complejidad: La estructura, el comportamiento y las permutaciones de los conjuntos de datos es un factor crítico
- Tecnologías: Las técnicas y herramientas utilizadas para procesar el inmenso y complejo conjunto de datos es también un factor clave

Las principales características que deben ser nombradas en big data son las denominadas "4 v" o "4 dimensiones" [6]

- Volumen: La información generada por las máquinas es producida en cantidades mucho más grandes que la que se produce por medios tradicionales. Por ejemplo, una simple turbina de avión es capaz de producir 10 terabytes de información en sólo 30 minutos. Si proyectamos eso a la cantidad de aviones existentes, con esa sola fuente de datos tendremos petabytes de información diariamente.
- Velocidad: El inmenso flujo de datos que producen por ejemplo los sitios sociales, producen una gran cantidad de opciones de entradas y de relaciones que son valiosas para la administración de los usuarios de dichos sitios, ya que en definitiva, agregan valor al usuario.
- Variedad: Los formatos tradicionales de datos tienden a ser bien definidos por un esquema de datos, y si tienen cambios, estos se producen esporádicamente, de manera lenta. Esto no ocurre con los formatos denominados "no-tradicionales" o "formatos big-data-compatibles", los cuales tienen una vertiginosa velocidad de cambio.
- Valor/Veracidad: El valor económico de los diferentes datos varía significativamente. Esto implica un desafío para quienes llevan la tarea del análisis de datos, pues existe mucha información que se considera "escondida" en la vasta información disponible. El desafío es identificar qué es lo que tiene valor, para extraerlo y transformarlo en información para ser analizada.

Existen además, diferentes tipos de datos con los que nos podemos encontrar en una plataforma de big data. Los mismos pueden diferir mucho, por lo que es conveniente clasificarlos de algún modo:

- Información "convencional" corporativa: Incluye desde información de los clientes en sistemas de tipo CRM [39], como también información sobre transacciones web (compras, solicitudes, facturación, etc.) e información contable generalizada.
- Información generada por máquinas o sensores
    - Información biométrica: Esta información incluye el almacenamiento de huellas digitales, patrones de retina, genética, reconocimiento facial. Generalmente esta información se guarda con fines de seguridad e inteligencia por parte de alguna entidad, por ejemplo agencias de seguridad de los diferentes estados.
    - Información generada por humanos: Como mencionamos anteriormente, un día de rutina en nuestra vida es capaz de generar muchísima información. Gran parte de esa información se categoriza en "generada por humanos". Aquí se pueden incluir desde correos electrónicos, documentos, llamadas telefónicas, conversaciones por mensajería instantánea, etc.
- Información social: Una de las categorías más crecientes de los últimos tiempos. En esta información se incluyen interacciones con las diferentes redes sociales como Facebook, LinkedIn, etc. Desde fotos, videos, artículos publicados en blogs. Toda información creada en una red social o similar pertenece a esta categoría.

Debemos entender de qué manera se pueden plantear los problemas para que puedan ser aplicados bajo esta filosofía. Cada actor que interviene en los diferentes procesos de big data debe tener bien definidas sus responsabilidades, dentro de su rol. En este sentido, desde el

campo de la informática y el desarrollo es fundamental conocer de qué manera resolver este tipo de problemas que pueden surgir del negocio o investigación.

4.2 Map Reduce

Desde nuestro rol como desarrolladores de software, es nuestra responsabilidad determinar cuál es el paradigma o modelo de programación a implementar para una solución determinada.

Dentro del campo de Big Data, es importante conocer y entender el modelo de programación denominado “map-reduce”, el cual se define como como un modelo de programación y su implementación asociada, para el procesamiento y la generación de grandes conjuntos de datos [5] .

Este paradigma define un estilo funcional de programación. Sus programas son automáticamente paralelizados y ejecutados en grandes clústeres de máquinas que no tienen requisitos exigentes a nivel hardware. Se las denomina “máquinas commodities”.

En tiempo de ejecución, el sistema se encarga de dividir la entrada de datos (proceso que se denomina “data partitioning”), planificar la ejecución de las diferentes tareas a lo largo del clúster, manejar los fallos y administrar la comunicación que se requiere entre las diferentes máquinas intervinientes del clúster.

Los usuarios especifican una función denominada “map” que procesa un par clave-valor para generar otro conjunto intermedio de pares clave-valor. Además se define otra función denominada “reduce” que une los valores intermedios asociados a una misma clave.

A continuación se define con mayor detalle cada una de estas etapas, las cuales conforman el modelo de programación de Map-Reduce:

- Map: Esta función, la cual es desarrollada por el usuario, toma un par de entrada (clave-valor) y produce un conjunto de pares clave-valor intermedios. La librería de map-reduce agrupa todos los valores asociados con la misma clave *I*, y la pasa a la función de reducción “reduce”. Se dice que esta función realiza un filtrado y/o ordenamiento de datos.
- Reduce: También desarrollada por el usuario, esta función acepta una clave intermedia *I* y un conjunto de valores asociados a esa clave. Luego une todos esos valores para entonces formar otro conjunto de valores, posiblemente más pequeño. Generalmente sucede que la función de “reduce” da como salida (output) cero o un valor. Los valores intermedios son proporcionados a la función de reducción a través de un iterador, lo cual nos permite manejar listas de valores más grandes que las que cabrían en memoria. Se dice que el proceso de reducción realiza una operación de “resumen”.

Desde el punto de vista lógico, tanto la función de map como la función de reduce se definen respecto de los datos estructurados en los pares clave-valor.

La función de map toma un par de datos de un dominio de datos, y retorna una lista de pares de un dominio diferente:

$$Map(k1,v1) \rightarrow list(k2,v2)$$

Esta función de Map se aplica en paralelo a cada par identificado por la clave *k1*, del conjunto de datos de entrada. Esto produce una lista de pares (identificados por la clave *k2*) por cada llamada. En este punto interviene el framework de MapReduce, quien colecta todos los pares de la misma clave *k2* de todas las listas y los agrupa, creando un grupo para cada clave.

La función de reducción se aplica luego en paralelo para cada grupo, y produce una colección de valores del mismo dominio:

$$Reduce(k2, list(v2)) \rightarrow list(v3)$$

Para dejar en claro en funcionamiento de este modelo de programación, se ilustra la definición con un ejemplo típico:

Consideremos el caso de querer contar la cantidad de ocurrencias de cada palabra en un gran documento. El pseudocódigo sería de la siguiente manera:

```
function map(String clave, String valor) {
      // clave: Nombre del documento
      // valor: contenido del documento
      for (w in valor) {
            emit(w, 1);
      }
}
```

```
function reduce(String clave, Iterator valores) {
      // clave: identificador de palabra
      // valor: lista de contadores asociados a esa clave
      int resultado = 0;
      for (v in valores) {
            result += v;
      }
      emit( clave, result);
}
```

En este ejemplo, la función de “map” emite cada palabra con un valor de ocurrencias asociado, que en este caso es 1 (uno). Luego la función de “reduce” suma todas las cantidades de ocurrencias para cada palabra en particular.

El gráfico 4.1 resume las diferentes fases con las que nos podemos encontrar en un proceso de map-reduce:

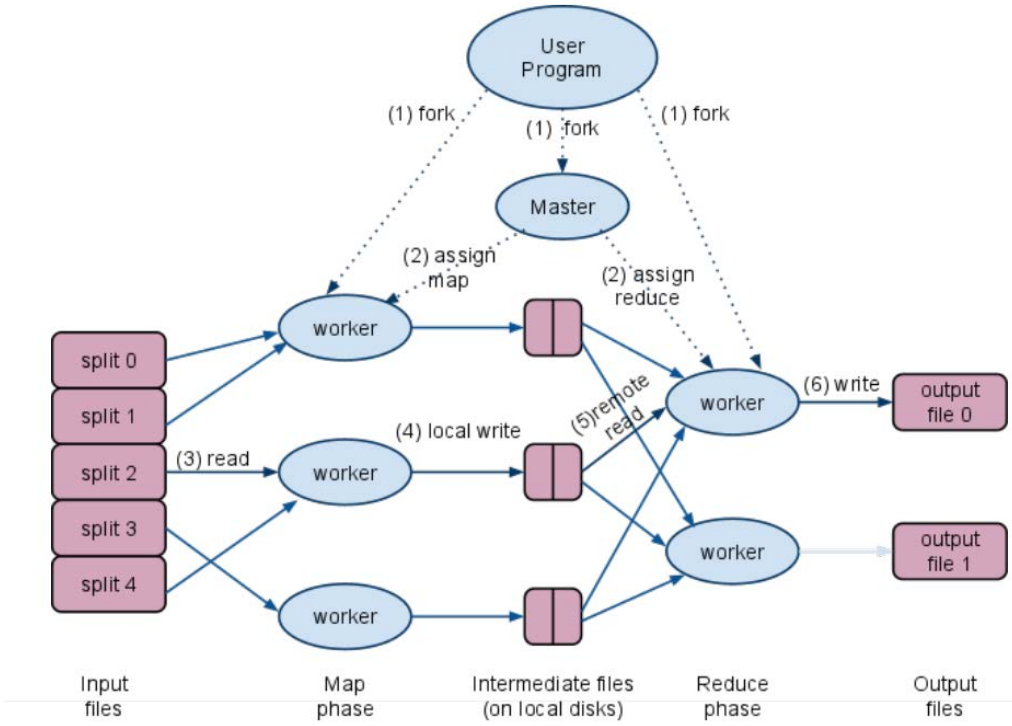


FIGURA 4.1 FASES EN UN PROCESO DE MAP REDUCE

Existen numerosos ejemplos donde se aplica perfectamente este modelo de programación, y que extienden al ejemplo básico presentado previamente:

- Contar frecuencia de acceso a URLs: Se podrían utilizar diferentes fuentes de datos, como los logs que producen los servidores web, para saber con qué frecuencia se accede a cada URL de nuestro servidor. La función de map leería los logs y produciría un par (url, 1) para que luego la función de reducer haga la suma correspondiente.
- Conocer el grafo reverso de links web: Este mecanismo nos permitirá conocer las fuentes de acceso a una web determinada. Aquí la función de map generaría, para cada link a una URL *target* que se encuentre en una web *source*, el par *(target, source)*. Luego, la función de reduce simplemente concatena todos los *sources* asociados a un mismo *target* : *(target, list(sources))* y los imprime en la salida.

En este capítulo nos hemos dedicado a definir, explicar, y ejemplificar el funcionamiento de Map Reduce. Sin embargo no es lo único modelo de programación con el que cuenta big-data. Sucede que Map Reduce es por lejos, el modelo preferido para la implementación de la mayoría de los algoritmos a ser ejecutados bajo big-data.

Fuera del contexto de Map Reduce, existen otros modelos de programación que se consideran partícipes de big-data. Dos ejemplos que han ganado popularidad en el último tiempo son:

- Pregel [40]: La entrada en un programa Pregel es un grafo dirigido en el cual cada vértice se identifica unívocamente por una cadena de caracteres llamado “identificador de vértice”. Cada vértice está asociado con un valor el cual es definido por el usuario y puede ser modificado. Las aristas dirigidas están asociados con un vértice fuente, y

constan de un valor el cual también es definido por el usuario y puede ser modificado; además constan también del identificador del vértice destino. Cuando el grafo está inicializado, será la entrada del programa Pregel, acompañado también de una secuencia de "super-pasos" separados por puntos de sincronización globales que se ejecutan hasta que el algoritmo termina, y finaliza con una salida. En cada "super-paso" de ejecución, los vértices se computan en paralelo y cada uno ejecuta la misma función definida por el usuario, la cual expresa la lógica de un determinado algoritmo. Con esto, un vértice puede modificar su estado o el de sus vecinos, puede recibir o enviar mensajes, o mutar la topología del grafo. La terminación de la ejecución se produce cuando todos los vértices votan por dicha opción, a través de un "halt". En el comienzo de la ejecución, todos los vértices se inicializan como activos, y durante la ejecución pueden ir cambiando su estado. Una vez que todos están inactivos, se produce la terminación. La salida de un programa Pregel es un con junto de valores de salidas explícitas producidas por los vértices.

- PACT [41]: El modelo de programación PACT (por su sigla del inglés *Parallelization Contracts*-Contratos de paralelización) es una generación de MapReduce, y se basa también en un modelo de datos del tipo clave-valor. El concepto clave de PACT son los contratos de paralelización (PACTs). Un PACT consta de exactamente una función de segundo orden llamada *Contrato de entrada,* y un *Contrato de salida* el cual es opcional. Un *Contrato de entrada* toma una función de primer orden la cual es código de usuario para ejecutar una tarea específica, y uno o más conjuntos de datos como parámetros de entrada. Es en este momento que el *Contrato de entrada* invoca a su función de primer orden con sub-conjuntos independientes de sus datos de entrada. Esto lo hace siguiendo un estilo de paralelismo de datos. Los *Contratos de salida* no son obligatorios, y es por esto que no tienen un impacto semántico en el resultado.

Capítulo 5 - HADOOP

Apache Hadoop, nacido en el año 2006, es un framework de software libre de la Apache Software Foundation - ASF [42] que permite el procesamiento distribuido de grandes cantidades de datos a lo largo de clústeres de computadoras. El hecho de que sea perteneciente a la ASF ha permitido un desarrollo sostenido por una comunidad más que activa, compuesta tanto por desarrolladores independientes, como también por organizaciones comerciales, como por ejemplo Yahoo!, quien es uno de los principales colaboradores.

El proyecto en sus comienzos era parte de Apache Nutch [43], también perteneciente a la Apache Software Foundation. Con el paso del tiempo fue adquiriendo importancia y participación, a tal punto de lograr independizarse.

La primera versión oficial fue la 0.1.0, liberada en Abril de 2006 y desde ese momento estuvo en constante evolución y adopción por parte de las principales compañías del mercado.

En el presente capítulo se nombran las principales características que hacen de este software el líder en big-data y MapReduce. Luego se sigue con la descripción de sus principales módulos: Hadoop Common, Hadoop Distributed File System (HDFS), Hadoop YARN y Hadoop MapReduce.

5.1 Características

Hadoop es un framework de código libre que permite el procesamiento distribuido de grandes cantidades de datos, utilizando modelos sencillos de programación, con MapReduce a la cabeza.

Su ejecución se lleva a cabo sobre un clúster de computadoras, permitiendo facilidad a la hora de necesitar escalar, pues sólo se necesitan agregar computadoras al clúster, y éstas comenzarán a ofrecer su capacidad de procesamiento y almacenamiento para participar en las tareas de Hadoop.

Su diseño de arquitectura está pensado para ejecutarse sin problemas en máquinas de bajo costo, lo que logra que sea un framework que cumple la definición de “commodity computing” [44] . Además cumple otra característica fundamental, y es que su diseño permite detectar y manejar fallas al nivel de la capa de aplicación, lo que permite ofrecer un servicio de alta disponibilidad, sin importar que las máquinas del clúster sean propensas a fallos. De esta manera no necesita computadoras de alta tecnología para garantizar la alta disponibilidad [45]. Esto lo convierte en un software tolerable a fallos.

Su desarrollo se realiza sobre el lenguaje de programación Java [46] lo que también ha facilitado la participación de la comunidad y la rápida evolución del framework, por ser Java uno de los lenguajes de programación más populares del mundo [47]

5.2 Arquitectura

Hadoop consta de un diseño de arquitectura que se divide en 4 grandes módulos, los cuales son:

- Hadoop Common
- Hadoop Distributed File System (HDFS)
- Hadoop YARN
- Hadoop MapReduce

Hadoop Common

Este módulo contiene una serie de librerías que hacen de soporte al resto de los 3 módulos. Es comportamiento compartido entre más de 1 módulo de Apache Hadoop. Incluye desde librerías del sistema operativo, como también funcionalidad extra, y utilitarios. Algunas librerías de este módulo que merecen ser nombradas:

- Hadoop CLI Mini Clúster: Se trata de un utilitario que permite ejecutar un clúster de Hadoop con sólo un comando, sin necesidad de configuraciones adicionales. Automáticamente ejecuta el módulo de YARN, HDFS y MapReduce. Es muy útil tanto para la ejecución de pruebas sencillas, como también para el momento inicial de aprendizaje. Permite ciertas configuraciones como cantidad de nodos para cada módulo de Hadoop.
- Librerías nativas de Hadoop: Hadoop incluye implementaciones nativas de algunas componentes, principalmente por cuestiones de performance, y también porque algunas no tienen implementaciones nativas en el lenguaje de programación Java. Todas estas componentes están disponibles bajo un único archivo, el cual bajo las plataformas Unix se denomina *libhadoop.so*. Entre las componentes que incluye se encuentran los codecs de compresión (bzip2 [48], lz4 [49], snappy [50], zlib [51]), utilidades nativas de entrada-salida para módulos del HDFS, y la implementación del checksum CRC32 [52]
- Hadoop en modo seguro: Por defecto, Hadoop no se ejecuta en modo seguro, por lo que no se requiere ningún tipo de autenticación. Configurando Hadoop para que se ejecute en modo seguro, cada usuario y servicio necesitará ser autenticado por Kerberos [53] para poder utilizar todos los servicios de Hadoop. Hadoop en modo seguro provee características de autenticación, autorización a nivel de servicio, autorización para las consolas Web, y confidencialidad de los datos.

Estas componentes de Hadoop Commons son las que, basados en la experiencia de esta tesina, merecen una mayor importancia, pero se debe saber que no son las únicas, y que Hadoop Commons está en un constante crecimiento y evolución.

HDFS

El sistema de archivos distribuido Hadoop (por sus siglas en inglés, HDFS), es un sistema de archivos distribuido diseñado para ser ejecutado en máquinas commoditiy. Posee diferencias significativas con otros sistemas de archivos distribuidos. Es altamente tolerante a fallos y se ejecuta en hardware de bajo costo. Provee un acceso de alto rendimiento a los datos de aplicación y es el sistema de archivos indicado para aplicaciones con grandes (o enormes) conjuntos de datos.

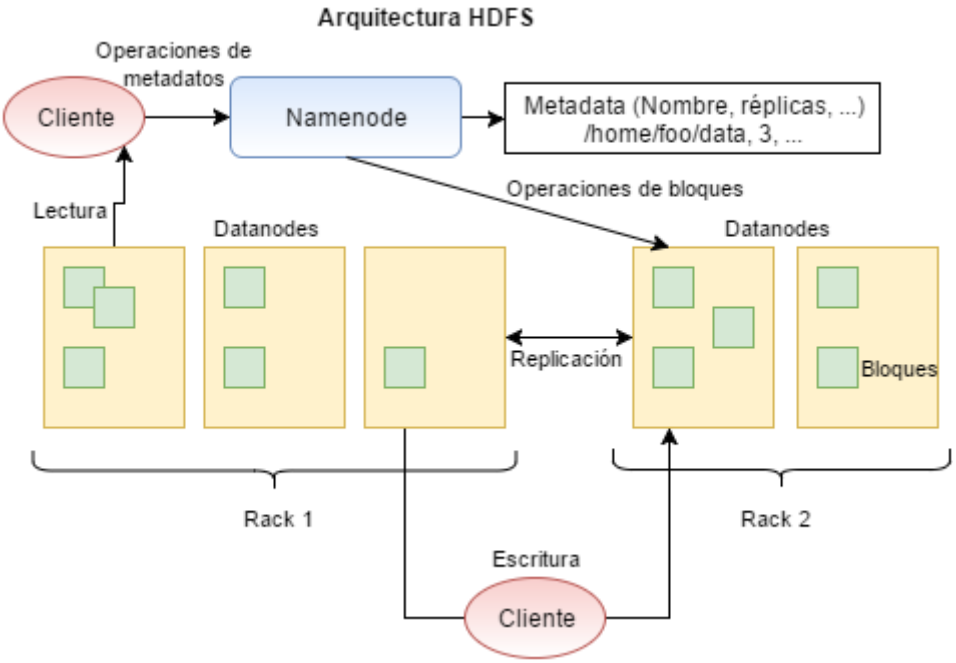


FIGURA 5.1 ARQUITECTURA GENERAL DEL SISTEMA DE ARCHIVOS DE APACHE HADOOP

HDFS tiene básicamente una arquitectura maestro/esclavo [54]

Un clúster de HDFS se forma por un NameNode, que cumple el rol de maestro, el cual administra la meta-información del clúster, y por DataNodes, que cumplen el rol de esclavos, y son quienes almacenan los datos. Generalmente existe un DataNode por cada nodo del clúster. Los archivos y directorios se encuentran representados en los NameNodes por “inodos”. Estos inodos llevan registro de los permisos, de la fechas de acceso y modificación, e información de los NameSpaces como espacio en disco, y cuotas.

Internamente, un archivo es dividido en bloques, los cuales son almacenados en un conjunto de DataNodes para asegurar redundancia y garantizar la tolerancia a fallos. El número de copias de un bloque es llamado el *factor de replicación* del bloque.

El NameNode ejecuta operaciones en el sistema de archivos como abrir, cerrar, y renombrar archivos y directorios. También se encarga de determinar el mapeo desde cada bloque hacia cada DataNode. El NameNode se encuentra constantemente monitoreando el estado de las réplicas de los bloques de archivo. Cuando una réplica de un bloque presenta un fallo, el NameNode crea otra réplica del bloque. De esta manera se evita pérdida de información.

El DataNode es responsable de atender las peticiones de lectura y escritura que provienen desde los clientes del sistema de archivos. También realiza la creación, eliminación y replicación de bloques de archivos a través de instrucciones desde el NameNode.

Una consideración de diseño importante, es que el NameNode no envía peticiones directamente hacia los DataNodes, sino que lo hace respondiendo los heartbeats (latidos del corazón) enviados por los DataNodes. Algunas instrucciones que puede enviar el NameNode: replicar bloque a otro nodo, remover réplica de bloque, enviar reporte de bloques, apagar nodo.

HDFS soporta el almacenamiento de archivos de forma jerárquica, es decir, como la conocemos comúnmente. Esto es, un sistema de archivos que consiste de directorios y archivos. Los

directorios al mismo tiempo pueden almacenar nuevamente archivos y/o directorios, y así sucesivamente.

Como se dijo anteriormente, la replicación de datos se realiza determinando el factor de replicación, el cual determina la cantidad de copias de los bloques de archivo que deben existir dentro del clúster de HDFS. El NameNode garantizará el correcto funcionamiento de estas réplicas, recibiendo heartbeats desde los diferentes NameNodes, los cuales incluirán reporte de estado global y de los bloques que almacena.

Cuando el NameNode recibe un heartbeat desde un DataNode, quiere decir que dicho DataNode se encuentra en línea. Además, en dicho heartbeat se envía la información de los bloques que contiene dicho DataNode.

Se explica a continuación el siguiente ejemplo:

FIGURA 5.2 ESQUEMA DE REPLICACIÓN DE BLOQUES EN EL SISTEMA DE ARCHIVOS DE APACHE HADOOP

En el gráfico 5.2 se observan dos archivos. Por su tamaño, el primero (part-0) generó 2 bloques de archivos identificados por (1) y (3). El segundo archivo (part-1) generó 3 bloques identificados por (2), (4) y (5).

Se observa que en este caso, cada archivo tuvo su configuración separada respecto del *factor de replicación*. Esta configuración se puede ajustar de manera global al clúster, como también por archivo. En el ejemplo, el archivo *part-0* tiene un factor de replicación 2, mientras que el archivo *part-1* tiene un factor de replicación 3.

Luego, en cada DataNode podemos ver los bloques almacenados, siempre de acuerdo a su configuración de factor de réplica. Como consecuencia, cada bloque será almacenado en "*n*" DataNodes, siendo *n* el factor de replicación del archivo.

Una herramienta fundamental que posee Hadoop es la posibilidad de tener "consciencia de rack" [55]. Esto implica que en un entorno de multi-rack, se tendrá en cuenta la disponibilidad de cada uno al momento de asignar los diferentes bloques de archivos.

Mientras recorremos las principales características del sistema de archivos distribuido de Hadoop, no hay que perder el foco de su principal objetivo: almacenar datos de manera fiable, incluso en la presencia de fallas. Existen 3 tipos de fallas comunes: fallas en el NameNode, fallas en el DataNode y particionamiento de la red. Con esto presente, podemos repasar una serie de

características y procesos que encontramos en un HDFS, y que juntos conforman el concepto de "robustez":

- Falla del disco de datos, Heartbeats y Re-Replicación: Los nodos pertenecientes a un clúster se comunican entre sí (DataNodes hacia NameNode) emitiendo un Heartbeat. Esto básicamente es un mensaje que indica el estado del DataNode, junto con otra información solicitada por el NameNode. Cuando nos encontramos en presencia de un error del tipo *particionamiento de red*, puede ocurrir que un DataNode no quede visible por el NameNode. Cuando el NameNode detecta esto (por ausencia de heartbeats) marca al nodo como fuera de línea, y deja de enviarle las diferentes solicitudes de escritura, lectura, etc. En este punto, toda la información contenida en el DataNode que se encuentra fuera línea, se considera inaccesible por el HDFS en su totalidad. Esto puede causar que sea necesario replicar nuevamente algunos bloques de archivo, para garantizar el factor de replicación y por consiguiente, la alta disponibilidad y tolerancia a fallos. Este proceso de re-replicación puede ser lanzado por varios motivos: uno de ellos, el ya mencionado, es cuando un DataNode se vuelve no-visible; pero además puede ocurrir que una réplica se vuelva corrupta, o que un disco de un DataNode falle.
- Rebalanceo del clúster: El diseño de arquitectura de HDFS permite tomar ciertas medidas para evitar el desbalanceo de los nodos del clúster. Esto sería básicamente mover datos de un DataNode a otro para evitar que en uno recaiga mayor carga que en el resto.
- Integridad de los datos: Como en cualquier almacenamiento, siempre existe la posibilidad de que los datos que obtengamos desde un DataNode estén corruptos. Esta situación se puede deber a muchos factores, pero principalmente se atribuye a fallas en la red, software con algún bug, o fallas en el DataNode. HDFS está diseñado para realizar la comprobación del checksum (recordar que incluye la librería CRC32) por cada bloque de archivo. Si la comprobación no es exitosa, el cliente puede optar por obtener el bloque de archivo desde otro DataNode, si existe replicación
- Falla del disco de meta-datos: Hasta este punto de la descripción del diseño de arquitectura, existe un único punto de fallo en todo el sistema de archivos HDFS, esto es el NameNode. En él, residen archivos que resultan de vital importancia para la vida del HDFS. Más adelante analizaremos algunas alternativas robustas y maduras para evitar este único punto de fallo. Una opción sencilla y fácilmente configurable, es realizar múltiples copias de los archivos núcleo del NameNode: FsImage y EditLog. De esta manera, cada vez que se realiza una operación en el NameNode, ésta se registra en todas las copias de los archivos FsImage y EditLog, de manera sincrónica. El costo que puede producir esta operación se compensa por la seguridad y tolerancia a fallos que se logra al implementar esta solución. Relacionado a esto, durante el proceso de arranque, HDFS se ejecuta en un modo que se considera "modo seguro" [56], lo que involucra la siguiente serie de pasos: conocer el estado del HDFS a partir de los archivos FsImage y EditLog. Luego esperar a que los DataNodes reporten sus bloques para saber cuáles son los bloques que cumplen con la condición de réplica (su factor de replicación). Durante este proceso se dice que el HDFS está en "modo seguro", que implica que el sistema de archivos estará disponible para sólo-lectura. Generalmente, el HDFS sale del "modo seguro" una vez que es consciente de que la mayor parte de sus bloques de archivo

están en buen estado, es decir, no están corruptos y hay suficientes réplicas de acuerdo al factor de replicación.

- Snapshots: HDFS permite realizar copias de un instante de tiempo en particular de los datos. De esta manera, en caso de existir errores, o de que se corrompan los datos, se puede volver a un punto en particular en el cual sepamos que el estado del HDFS era óptimo. El funcionamiento es similar al *punto de restauración*, existente por ejemplo, en los sistemas operativos de la familia Windows [57]

Otras componentes de la arquitectura de Hadoop en el HDFS:

NameNode Secundario

Para entender el funcionamiento y la necesidad que puede existir de tener un NameNode secundario, debemos tener en claro en primer lugar cómo funciona el NameNode.

Cada modificación que se realiza en el sistema de archivos HDFS se almacena en el NameNode en un archivo de log llamado "*edits*", el cual se encuentra en el sistema de archivos nativo. Cuando HDFS comienza a funcionar, se encarga de leer su propio estado desde el archivo FsImage y luego aplica los cambios que se encuentran en el archivo "*edits*". Una vez hecho esto, HDFS crea un nuevo archivo FsImage, con el estado actual del sistema de archivos. Con esto hecho se encuentra en condiciones de vaciar el archivo "*edits*". Esta operación solo se realiza durante el arranque y puesta en marcha del HDFS. Por esta razón, puede suceder que el archivo "*edits*" se torne demasiado grande, además de que la operación de juntar los archivos FsImage y *edits* puede ir volviéndose costosa con el paso del tiempo.

Para evitar este escenario es que entra en juego el NameNode secundario, el cual suele ejecutarse en una máquina diferente ya que suele tener requisitos de memoria que pueden igualar al NameNode primario. Tendrá la tarea de realizar la combinación de los archivos FsImage y *edits* de manera periódica, y también de mantener al archivo *edits* con el menor tamaño posible.

Nodo checkpoint

Siguiendo en la línea con el NameNode secundario, el Nodo Checkpoint tiene un papel similar. Este nodo se encarga de crear periódicamente "checkpoints" del namespace, el cual consta de los archivos FsImage y *edits*. La tarea consiste en descargar dichos archivos del NameNode activo, unirlos localmente y subir la nueva imagen del namespace al NameNode activo. El nodo checkpoint también suele ejecutarse en una máquina diferente a la del NameNode por los requerimientos de memoria que suele involucrar.

Nodo backup

Este nodo tiene una responsabilidad parecida a la del Nodo Checkpoint, pero también mantiene una copia del namespace del sistema de archivos en memoria, la cual está constantemente sincronizada con el namespace del NameNode activo.

Para lograr este objetivo, el nodo backup recibe el stream de instrucciones, las cuales réplica en su disco y también en su copia en memoria.

A diferencia del NameNode secundario o el Nodo Checkpoint, el Nodo Backup no requiere descargarse los archivos FsImage y *edits* desde el NameNode activo ya que constantemente tiene una copia fiel del namespace en memoria.

Ya que el principal almacenamiento lo realiza sobre memoria RAM, los requerimientos de ésta suelen ser iguales a los del NameNode activo.

El NameNode soporta el uso de un nodo backup por vez. Además, si existe un Nodo backup, no hay necesidad de tener un Nodo Checkpoint.

HDFS Federación

Hasta el momento hemos profundizado sobre las características que nos proporciona el sistema de archivos de Hadoop, HDFS. Sin embargo, debemos notar una limitación en toda la descripción previamente realizada, que es clave para nuestro objetivo de alta disponibilidad y tolerancia a fallos.

Puntualmente, el NameNode es nuestro punto de falla. Las razones son obvias, ya que el NameNode consta de un único nodo, con lo que a la menor falla de hardware o software, todo el sistema de archivos HDFS se verá afectado. Pero también existe otra limitación, y es que HDFS mantiene su namespace en memoria RAM, con lo que limita la cantidad de archivos que se podrían almacenar, sin importar cuánto espacio de almacenamiento tengamos en los diferentes DataNodes.

Con el objetivo de resolver estos inconvenientes es que surge HDFS Federación [58]. Este mecanismo nos permite escalar horizontalmente [59] nuestro NameNode y por consecuencia, nuestro namespace, agregando más nodos que actúan como NameNodes. Todos los NameNodes del clúster son independientes entre sí, lo que justifica el nombre de “Federación”. Se dice que los NameNodes son federados, pues no se requiere ni siquiera coordinación entre ellos. Todo el almacenamiento disponible, formado por los DataNodes, es visto como un almacenamiento común para poder ser utilizado por los diferentes NameNodes.

La única modificación que tienen que manejar los DataNodes es que ahora, en vez de comunicarse con un sólo NameNode, lo harán con todos los que intervengan en el clúster.

Existe dentro de HDFS Federación el concepto de ***pool de bloques***. Esta estructura la encontramos en cada NameNode, y contiene el conjunto de bloques que pertenecen al namespace de dicho NameNode. Cada pool de bloques del clúster es administrado de manera independiente. De esta manera, ante una falla en un NameNode, todos los DataNodes pueden seguir atendiendo peticiones de los restantes NameNodes que sigan en línea.

HDFS Federación nos trae un nuevo concepto llamado *Volumen del Namespace* el cual se conforma en conjunto por el Namespace propiamente dicho y su pool de bloques. Este nuevo concepto es una unidad de administración auto-contenida: Como consecuencia, si un NameNode o su namespace se borra, también lo hará su correspondiente pool de bloques, y todos sus datos en los diferentes DataNodes intervinientes.

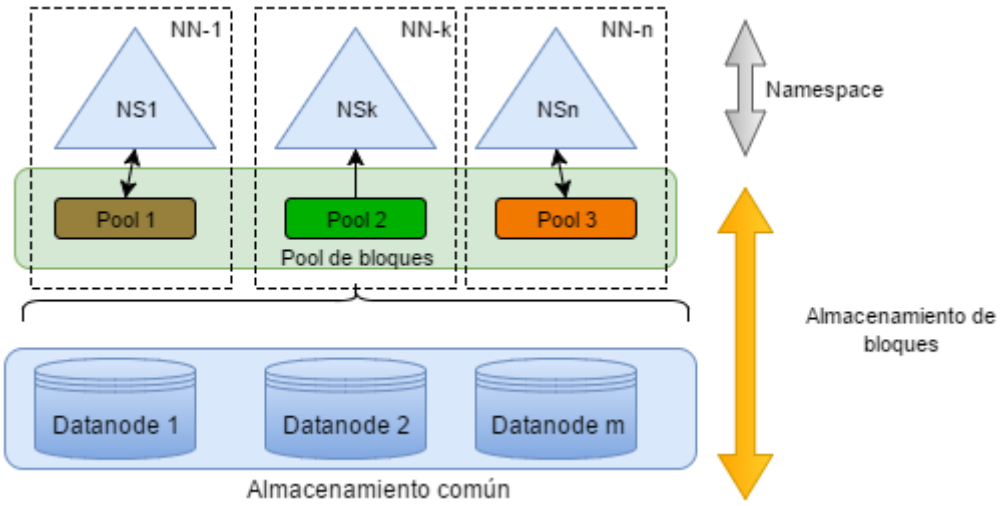


FIGURA 5.3 ARQUITECTURA DEL ESQUEMA DE FEDERACIÓN DENTRO DE HADOOP

En consecuencia, estos son los principales beneficios de HDFS Federación:

- Escalabilidad del NameNode: La funcionalidad de HDFS Federación permite escalar horizontalmente al NameNode, superando una de las limitaciones existentes en HDFS
- Mejor performance: Esto se produce directamente al tener más nodos, por el concepto de escalabilidad. Al tener más NameNodes, podremos realizar mayor cantidad de lecturas y escrituras.
- Aislamiento (Isolation): Sin HDFS Federación, teníamos un único punto de acceso al sistema de archivos, sin importar de qué o quién era el que estaba queriendo acceder. Con múltiples NameNodes, se pueden tener segmentaciones y asignar diferentes NameNodes a cada segmento (por ejemplo de uso intensivo para lectura, uso intensivo para escritura, datos transaccionales, etc.)

Esta funcionalidad ha sido fundamental en el camino de maduración de Apache Hadoop y le ha provisto de enorme robustez.

YARN

Apache YARN (por sus siglas del inglés: Yet Another Resource Negotiator) es el sistema de administración de recursos a nivel clúster que provee Hadoop.

En las primeras versiones de Hadoop, este módulo no existía, lo que implicaba una limitación a la hora de implementar trabajos distribuidos, pues Hadoop sólo podía ejecutar programas bajo el paradigma de MapReduce. Con Hadoop 2.x esto queda en el pasado, pues YARN es lo suficientemente genérico como para soportar otros paradigmas de cómputo distribuido. De esta manera, *MapReduce*, cuya implementación en Apache Hadoop será explicada más adelante, pasa a ser uno de los múltiples paradigmas de programación disponibles en YARN, junto con otros como MPI, procesamiento de grafos, etc.

Es necesario dar una revisión por la arquitectura que brindaba Apache Hadoop 1.x, plasmada en la figura 5.4, para entender las ventajas y evolución que nos provee el nuevo módulo de Apache Hadoop YARN:

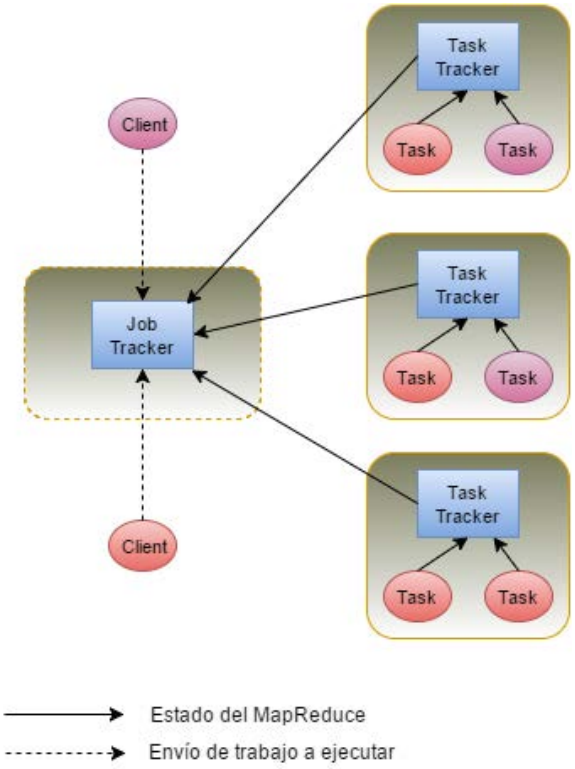


FIGURA 5.4 ESQUEMA DE EJECUCIÓN EN APACHE HADOOP 1.X

Como dijimos, el único paradigma permitido era el de MapReduce, el cual constaba de un JobTracker, ubicado en el nodo master, y de diferentes Task Tracker, ubicados en los diferentes nodos esclavos.

El JobTracker tenía responsabilidad de la administración de recursos, (esto lo hacían administrando los nodos denominados “workers”, es decir, los que contenían TaskTrackers), llevando registro de los recursos que se consumían, y aquellos que estaban disponibles. También tenía la responsabilidad de administrar el ciclo de vida de una ejecución de trabajo.

El TaskTracker tenía responsabilidades sencillas, como lanzar o apagar tareas bajo la orden del JobTracker, y también informar a éste con información de estado de las tareas en ejecución.

Con esta arquitectura, el JobTracker debía considerar varios aspectos relacionados a la escalabilidad y la utilización de recursos del clúster.

Las limitaciones fueron surgiendo. Además de lo mencionado anteriormente, la limitación de ejecutar sólo programas bajo el modelo de MapReduce fue creciendo ante la necesidad de ejecución de aplicaciones del tipo de tiempo real.

Con este escenario planteado, entenderemos de manera clara la necesidad de evolucionar hacia un administrador de recursos más genérico y potente.

En este sentido, la idea fundamental de YARN es dividir en dos las responsabilidades del JobTracker: administración de recursos y planificación/monitoreo de tareas. Esto se logra con dos componentes del tipo “demonio” o “daemon” [60]: por un lado el ResourceManager global, y por otro, el ApplicationMaster, que funciona por aplicación.

El ResourceManager es global, pero consta de múltiples NodeManager que residen en cada uno de los nodos esclavos, y juntos forman el nuevo y genérico sistema para la administración de aplicaciones distribuidas.

El ResourceManager es la máxima autoridad a la hora de arbitrar los recursos disponibles en el clúster entre todas las aplicaciones del sistema. El ApplicationMaster (uno por aplicación) es una entidad de algún framework específico y su principal tarea es negociar recursos del ResourceManager y trabajar con los NodeManagers para ejecutar y monitorear las diferentes tareas.

El ResourceManager tiene un planificador (scheduler) el cual se puede intercambiar de acuerdo a las diferentes necesidades. La principal responsabilidad de este planificador es la de alocar recursos para las aplicaciones de acuerdo a las restricciones que implemente. Este planificador es puro en su definición, pues no realiza tareas de monitoreo ni conoce el estado de la aplicación, por lo que no ofrece ninguna garantía de re-iniciar tareas que hayan fallado ni por problemas de hardware, ni por problemas de aplicación.

El NodeManager, que se encuentra en cada nodo esclavo, es el responsable de ejecutar los diferentes contenedores de la aplicación, monitoreando el uso de sus recursos (CPU, memoria, disco, red) y reportando esto al ResourceManager.

El ApplicationMaster (uno por aplicación) tiene la responsabilidad de negociar los contenedores de recursos apropiados desde el planificador, conociendo sus estados y monitoreando progresos. El ApplicationMaster se ejecuta también como un contenedor.

Podemos ilustrar el funcionamiento de Apache Hadoop YARN de la siguiente manera:

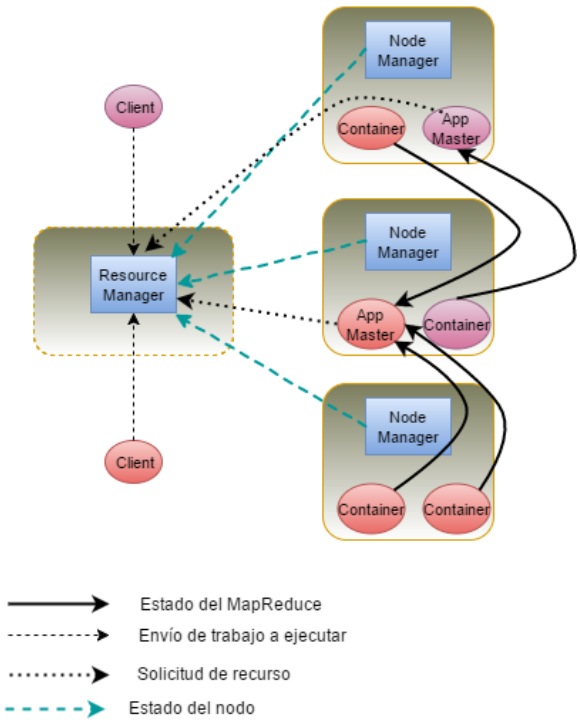


FIGURA 5.5 ESQUEMA DE FUNCIONAMIENTO EN APACHE HADOOP YARN

Como dijimos, el ApplicationMaster es responsable de solicitar recursos al ResourceManager en caso de necesitarlos. Esto se hace a través del denominado ResourceRequest, el cual tiene el siguiente formato:

<nombre-recurso, prioridad, requerimiento-del-recurso, número--de-contenedores>

Se describe cada componente del ResourceRequest:

- Nombre del recurso: Hace referencia al nombre del host, o del rack. Puede ser "*" para indicar que no hay preferencia
- Prioridad: Se refiere a una prioridad del recurso intra-aplicación. Es decir, no se maneja una prioridad entre múltiples aplicaciones.
- Requerimiento del recurso: Explícitamente se indica el recurso que se requiere. Hoy se soporta CPU y Memoria
- Número de contenedores: Es un múltiplo de cuántos contenedores voy a solicitar con la información detallada en los otros atributos.

Esencialmente, el Contendor es la representación de un recurso correctamente alocado.

Cuando el ApplicationMaster recibe la respuesta a su ResourceRequest (es decir el o los Contenedores), debe presentar dicho contenedor al NodeManager del nodo al cual fue asignado, para que de esta manera, los recursos puedan ser utilizados por la tarea específica.

Aclarado este concepto, podemos decir que un Contenedor no es más que el derecho a usar una determinada cantidad de recursos en una máquina específica del clúster. Sin embargo, esto no alcanza para ejecutar una tarea. El ApplicationMaster debe proveer más información al NodeManager para realmente ejecutar el Contenedor.

Los Contenedores tienen una API de especificación que es agnóstica a la plataforma y que contiene los siguientes parámetros:

- Línea de comandos para ejecutar el proceso dentro del contenedor
- Variables de ambiente
- Recursos locales necesarios en la máquina, previo a la ejecución, como archivos JARs, objetos compartidos, archivos de datos auxiliares, entre otros.
- Diferentes Tokens requeridos por seguridad.

Para entender de manera clara qué pasa cuando se ejecuta una aplicación en Apache Hadoop YARN, se enumeran a continuación los pasos que intervienen en el proceso:

1. Un programa que actúa como cliente de YARN envía la solicitud para ejecutar la aplicación, incluyendo además toda la especificación necesaria para lanzar el ApplicationMaster de dicha aplicación
2. El ResourceManager asume la responsabilidad de negociar el Contenedor que se necesita para iniciar el ApplicationMaster. Una vez que lo obtiene, lo ejecuta.
3. El ApplicationMaster, durante su inicio, se registra con el ResourceManager.
4. Durante el normal funcionamiento, el ApplicationMaster negocia los contenedores de recursos apropiados a través del protocolo de ResourceRequest
5. En caso de lograr una alocación exitosa de contenedores, el ApplicationMaster lanza el contenedor, junto con todos los datos necesarios, en el NodeManager seleccionado.
6. El código de aplicación se ejecuta en el contenedor, y éste brinda información de progreso, estado, etc., a su ApplicationMaster a través del protocolo de aplicación específico
7. En el transcurso de la ejecución de la aplicación, el cliente que mandó a ejecutar el programa se comunica directamente con el ApplicationMaster para obtener su estado, actualizaciones del progreso, etc. Esto lo logra a través del protocolo de aplicación específico
8. Una vez que el programa se completa, y todo el trabajo necesario ha finalizado, el ApplicationMaster se des-registra del ResourceManager y se apaga, permitiendo que se liberen todos los recursos de los Contenedores involucrados.

Esta serie de pasos se ilustra en la figura 5.6:

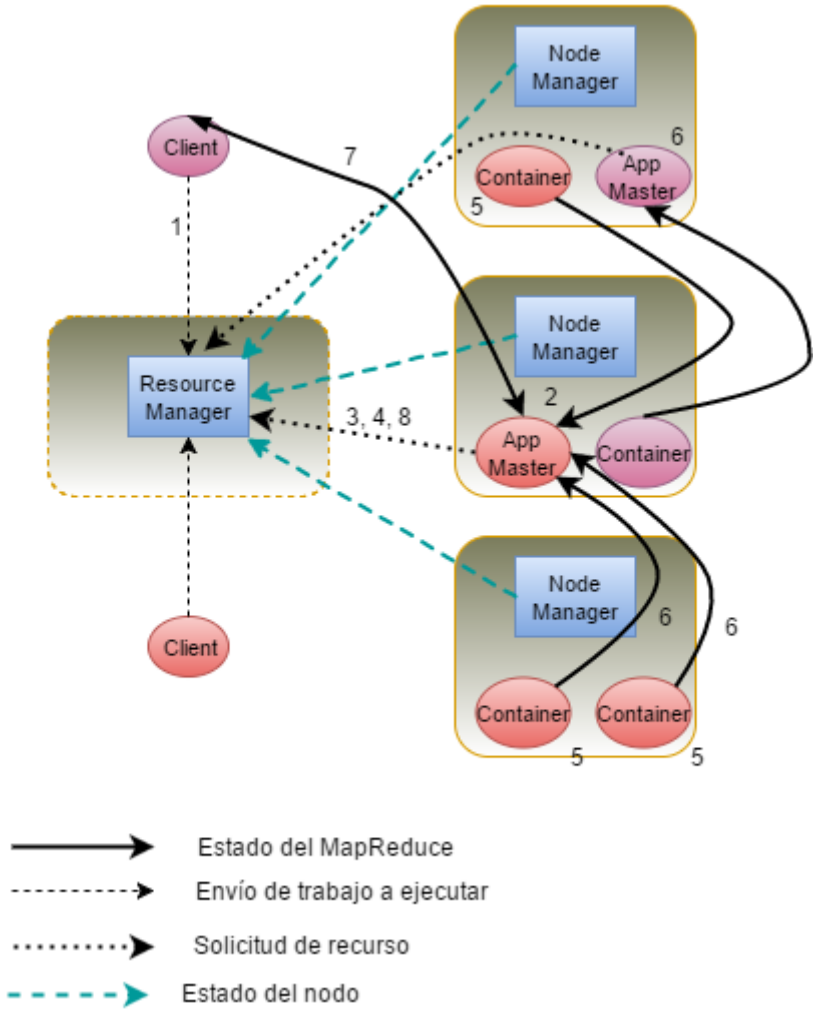


FIGURA 5.6 PASOS EN LA EJECUCIÓN DE UN TRABAJO EN YARN

Esta descripción completa nos ha permitido comprender la arquitectura y el funcionamiento de YARN. Se ha realizado un recorrido por sus componentes, identificándolas y haciendo una descripción de las responsabilidades que a cada una le corresponden.

Vale la pena mencionar que cada componente de YARN cuenta con una interfaz de aplicación, es decir, una API para ser consultada y operar. Esto facilita algunas tareas tanto del programador como también del administrador de sistemas.

La principal componente que cuenta con esta funcionalidad es el ResourceManager, a través de su ResoureManager REST API [61]. También podemos encontrar esta funcionalidad en el NodeManager a través de la NodeManager REST API [62].

MapReduce

La última componente que nombraremos en este recorrido sobre Apache Hadoop, es la correspondiente al framework de MapReduce.

En su definición, Hadoop MapReduce es un framework para facilitar el desarrollo de aplicaciones que procesan vastas cantidades de datos en paralelo sobre un clúster de miles de nodos, con la

característica de ser, cada nodo, una máquina commodity. Esta definición se encuentra ampliada en un apartado dedicado a MapReduce dentro del capítulo de BigData.

Este framework de MapReduce consiste de un sólo ResourceManager que reside en el master, y de un NodeManager por cada nodo del clúster. Por cada aplicación consta de un MRAppMaster, el cual es la implementación específica del ApplicationMaster de MapReduce.

Mínimamente, este tipo de aplicaciones especifican las ubicaciones de la fuente de entrada y de salida. Además se proporcionan las funciones de *map* y de *reduce* a través de las correspondientes implementaciones de interfaces o clases abstractas. Todo esto, junto con otros parámetros del trabajo, compone lo que se denomina la *Configuración del Trabajo*.

El funcionamiento para su ejecución, luego es de igual manera que para cualquier otra aplicación que se ejecute en Apache Hadoop YARN.

Para los desarrolladores y administradores de sistemas se encuentra disponible una API de MapReduce ApplicationMaster [63] que facilita el acceso a cierta información como su estado, sus tareas, contadores, configuraciones, etc.

A continuación se enumeran y describen las principales interfaces que intervienen en un programa map-reduce, y que deben ser implementadas por el desarrollador para lograr el correcto funcionamiento:

- Mapper: Los Maps son tareas individuales que transforman registros de entrada en registros intermedios. Los registros intermedios transformados no necesariamente son del mismo tipo que los registros de entrada. Un registro de entrada puede producir cero o varios pares de salida. Hadoop MapReduce genera una tarea de Map por cada *InputSplit* generado por el *InputFormat*, el cual se describe en breve. Al trabajo a ejecutar le debemos indicar la implementación del Map a utilizar con el método *Job.setMapperClass(Class)*. Todos los valores que se producen de salida, y que están asociados a una misma clave pueden ser agrupados previamente a ser pasados al Reducer. Esta agrupación se logra implementando la interfaz *Comparator*, e indicándole dicha implementación a la configuración de la ejecución a través del método *Job.setGroupingComparatorClass(Class)*. Todas las salidas intermedias son ordenadas y luego particionadas para ser enviadas al Reducer. Por consecuencia, la cantidad de particiones determinará la cantidad de Reducers. Se puede controlar cuáles claves van con cada Reducer implementando la interfaz *Partitioner*. Los usuarios pueden opcionalmente indicar un “combinador”, lo que permite realizar agregaciones locales de las salidas intermedias, achicando de esta manera la cantidad de datos que se transfieren y procesan en el Reducer.
- Reducer: Esta interfaz se implementa con el objetivo de dar comportamiento a la fase de reducción de un trabajo de MapReduce. Reduce un conjunto de valores intermedios asociados a una misma clave. Se puede configurar el número de reducers del trabajo a través del método *Job.setNumReduceTask(int)*. Para indicar al trabajo la implementación de Reducer a utilizar, se debe realizar a través del método *Job.setReducerClass(Class).* La interfaz de Reducer consta de 3 fases, las cuales se describen a continuación:
    - Shuffle: En esta fase, el framework obtiene la partición relevante de la salida de los mappers, a través del protocolo HTTP

    - Sort: En esta fase, el framework agrupa las claves de las entradas de los Reducers (recordemos que diferentes mappers pueden dar como resultado mismas claves). Esta etapa y la de shuffle ocurren en simultáneo, a medida que se va obteniendo la salida de la etapa de Mapper
    - Reduce: En esta fase se ejecuta el método *reduce(WriteComparable, Iterable<Wirtable>, Context)* por cada par *<clave, (lista de valores)>* agrupado como entrada. La salida de esta fase generalmente se graba en el *FileSystem* [64] a través del método *Context.write(WritableComparable, Writable)*. Es importante mencionar que salidas del Reducer no se encuentran ordenadas bajo ningún criterio.
- Partitioner: Esta interfaz se encarga de especificar cómo particionar el espacio de claves intermedias que se obtienen como resultado del Mapper. Una clave (o un subconjunto de ellas) deriva en una partición, generalmente a través de una función de hash. El número total de particiones determina el número de tareas de reducción. Existe una implementación por defecto del Partitioner, llamada HashPartitioner [65]
- Counter: Esta interfaz es una utilidad para que las aplicaciones MapReduce reporten sus estadísticas. La interfaz puede ser usada por las implementaciones de Mapper y Reducer.
- InputFormat: Esta interfaz es la especificación de la entrada para un trabajo de MapReduce. La interfaz cumple un rol muy importante a la hora de validar la especificación de entrada para un determinado trabajo. También es importante en el momento de dividir los archivos de entrada y generar los *InputSplit*, es decir, la representación de datos a ser procesados por un Mapper en particular [66]. Por último, el *InputFormat* es muy importante para que el *RecordReader* [67], que se encarga de leer cada par clave-valor desde un *InputSplit*, sepa cómo tratar cada par a ser procesado por el Mapper.
- OutputFormat: La interfaz *OutputFormat* describe la especificación de salida para un trabajo de MapReduce. Esto es importante para validar la especificación de salida de un trabajo (por ejemplo, comprobar que el directorio de salida de un trabajo ya está creado o no). También sirve para proveer a la implementación del *RecordWriter* [68], interfaz que escribe el par de salida <clave, valor> a un archivo de salida, generalmente en el *FileSystem*.

Conociendo estas interfaces, estamos en condiciones de escribir nuestra propia aplicación de MapReduce para ser ejecutada en un entorno de Apache Hadoop YARN.

Como hemos visto, las posibilidades son enormes. No debemos perder de vista, no sólo las oportunidades que nos brinda el framework de Apache Hadoop, sino la flexibilidad y personalización que exponen a los diferentes actores que se involucran en el desarrollo y despliegue de una aplicación de estas características. Desde el administrador de servidores, hasta el programador, contarán con las herramientas necesarias para adaptar cada trabajo a su necesidad de negocio.

5.3 Apache Hadoop - Soluciones Cloud

Los dos líderes en servicios de cloud computing, Amazon y Microsoft, ofrecen cada uno una solución para el desarrollo ágil de aplicaciones que se ejecuten sobre Apache Hadoop.

Estas soluciones tienen el foco puesto no sólo en facilitar el desarrollo y despliegue de aplicaciones, sino principalmente en la optimización de costos.

La puesta en marcha de un clúster de Apache Hadoop, junto con la estimación de capacidad, puede tornarse una tarea para nada fácil. Errores de estimación nos pueden llevar a asumir costos que luego la compañía no pueda afrontar. De igual manera, el tiempo que insuma la puesta en marcha o administración del clúster puede generar un problema a la hora de desplegar nuestro trabajo de Apache Hadoop.

Estas soluciones son una alternativa a la instalación manual y convencional de un clúster Hadoop, y bajo ningún concepto implican un reemplazo a esa opción.

Haciendo el foco con estos fundamentos, resulta conveniente conocer estas soluciones de mercado y contemplarlas a la hora de empezar un nuevo desarrollo:

Microsoft Azure HDInsight

El servicio de HDInsight es un tipo de implementación de Hadoop que se ejecuta en la plataforma de Microsoft Azure. Su desarrollo se realizó en base la plataforma de datos de Hortonworks (Hortonworks Data Platform - HDP), lo que lo hace 100% compatible con Apache Hadoop.

HDInsights es una alternativa a la instalación manual de un clúster de Apache Hadoop. Algunas ventajas de su uso son:

- Se puede desplegar rápidamente el sistema desde el portal de Microsoft Azure, o a través de la consola de Windows PowerShell, sin tener que crear ninguna máquina virtual ni física.
- Se pueden crear clúster de pocas máquinas o también de muchas.
- Sólo se paga por lo que se usa.
- Cuando un trabajo finaliza, se puede deprovisionar el clúster, permitiendo mayor optimización en costos.
- Se puede combinar su uso con el almacenamiento que ofrece Microsoft Azure Storage. De esta manera, cuando un trabajo finaliza y el clúster se deprovisiona, los datos aún quedan disponibles

Amazon Elastic MapReduce

El servicio de Amazon Elastic MapReduce (Amazon EMR) facilita el procesamiento rápido y rentable de grandes volúmenes de datos. Su principal foco está puesto es simplificar el procesamiento en el campo de big data, trabajando con el ecosistema de Apache Hadoop.

Esta ejecución se apoya sobre las instancias de Amazon EC2 [69], donde se puede escalar de manera sencilla y económica.

Por defecto, Amazon EMR instala y configura por lo menos las siguientes componentes: Hadoop MapReduce, YARN y HDFS. Esto lo hace en todos los nodos del clúster. Adicionalmente se

pueden instalar aplicaciones como Hive o Pig. Algunas ventajas que nos pueden llevar a esta elección:

- Velocidad y agilidad: Se gana velocidad a la hora de iniciar un nuevo clúster de Amazon EMR. Además, agregar o eliminar nodos a un clúster existente se realiza de manera ágil y segura.
- Reducción de complejidad administrativa: Amazon EMR contempla la mayoría de las configuraciones de infraestructura necesarias para operar con Apache Hadoop, por ejemplo: configuraciones generales, redes, instalación de Hadoop, entre otros.
- Integración con otros servicios en la nube: Se integra fácilmente con el resto de la familia de productos ofrecidos por Amazon, como por ejemplo Amazon S3, Amazon DynamoDB, Amazon Redshift, entre otros.
- Costos: Se mejoran notablemente los costos, pues existe la elasticidad. Esto nos va a permitir agregar recursos cuando sean necesarios, y apagarlos cuando estén ociosos. Así se podrán soportar los eventuales picos de trabajo sin tener un costo fijo elevado.
- Disponibilidad y recuperación: Haciendo uso de las zonas de disponibilidad que ofrece Amazon [70], se garantiza la alta disponibilidad y la tolerancia a fallos.

Apache Hadoop nos ofrece un mundo de posibilidades a la hora de trabajar con procesos y datos, es por eso que resulta fundamental conocer sus posibilidades y herramientas, pues un programador o arquitecto de software debe tomar la decisión acorde a sus necesidades, y que también acompañen las exigencias del negocio.

En este capítulo se ha mostrado la gran capacidad que nos ofrece el software Apache Hadoop. Se realizó una descripción que no sólo sirve para reforzar los conceptos de Big Data, Map Reduce, y Cloud Computing, sino que también nos servirá para comenzar con el desarrollo de nuestra herramienta de trabajo.

Capítulo 6 - HERRAMIENTAS DESARROLLADAS

En este capítulo se presenta el desarrollo de dos aplicaciones, las cuales persiguen los siguientes objetivos:

- Llevar al usuario final la posibilidad de ejecutar trabajos en YARN y conocer el posterior estado del mismo: Para esto se desarrolla una aplicación capaz de lanzar la ejecución de un trabajo bajo el framework YARN, de manera simplificada. La aplicación posibilita también el seguimiento de los diferentes trabajos en ejecución.
- Facilitar la interacción con el sistema de archivos de Hadoop (HDFS), eliminando la complejidad que puedan presentar los clientes y las APIs existentes. Para esto el desarrollo propone una interfaz para interactuar de manera amigable con el sistema de archivos de Hadoop, utilizando comandos y estructuras familiares para cualquier persona que haya interactuado en otro sistema de archivos en un sistema operativo como Linux o Windows.
- Ambos desarrollos surgen de la necesidad de demostrar la posibilidad de ejecutar servicios en un ambiente de computación en la nube, utilizando una interfaz REST, y sirven principalmente a este objetivo.

El capítulo comienza nombrando y describiendo las tecnologías involucradas en los desarrollos. Luego se continúa con la descripción de cada una de las aplicaciones, explicando las funcionalidades que exponen al usuario. Para finalizar el capítulo, se hace un repaso sobre las herramientas similares a las implementadas que existen actualmente en el mercado

6.1 Tecnologías involucradas

Ambas aplicaciones fueron desarrolladas bajo el lenguaje de programación Java [71], un lenguaje orientado a objetos, derivado y familiar a C, característica que hace que sus sintaxis sean similares, facilitando de esta manera el aprendizaje para quienes tengan experiencia en un lenguaje de dicha familia. La esencia de Java es la JVM –del inglés, Java Virtual Machine, componente que se encarga de la ejecución de un programa.

El código Java se programa en archivos de extensión '.java' los cuales se compilan en archivos de extensión '.class', compuestos de *bytecodes*, es decir, instrucciones que la JVM puede interpretar [72]. A la hora de ejecutarse, estos archivos son interpretados por la JVM. Esta característica hace que los *bytecodes* se comparen con instrucciones que se ejecutan sobre un procesador (en este caso el procesador sería la JVM).

Gracias a la existencia de la JVM, el código Java es portable, pues sus instrucciones no varían entre diferentes arquitecturas de hardware. El único requisito para ejecutar un programa Java, es contar con la JVM adecuada para nuestra arquitectura.

Una característica que ayudó también a la elección del lenguaje fue el amplio soporte existente en la comunidad Java, la cual está muy activa y con muchos años de maduración. Existen también un sinnúmero de librerías y complementos que ayudan y acompañan a la tarea del programador.

Para el desarrollo de ambas aplicaciones se utilizó el entorno de desarrollo integrado [73] por excelencia de Java, Eclipse [74]. Este software ayuda al programador en todo el ciclo de vida del desarrollo del software. Desde la creación del proyecto, guía y marca errores de sintaxis, ofreciendo en la mayoría de los casos una posible solución. También permite ejecutar y realizar debug de la aplicación. Permite la integración con muchas herramientas como por ejemplo versionadores de código, manejadores de dependencias, automatizadores, entre otros.

El desarrollo de estas aplicaciones fue posible por la utilización de otras librerías, es decir, piezas de software que podemos incluir en nuestro desarrollo. Para administrar estas librerías se utiliza Maven [75], que no sólo es un manejador de dependencias, sino mucho más. Se lo considera una herramienta para la administración y construcción del proyecto. Consta de diversas tareas y plug-ins para ser utilizados al momento de generar el código fuente y empacar la aplicación, dejándola lista para ser desplegada. Eclipse cuenta con un plug-in que integra con Maven, aunque también puede ser utilizado directamente desde la línea de comandos.

El núcleo principal de ambas aplicaciones se basa en el framework de desarrollo Spring [76]. La elección se justifica en varios puntos, aunque los principales motivos incluyen a una comunidad activa, gran soporte, listas de distribución, y por supuesto, que es un software muy robusto que resuelve muchas de las problemáticas que se planteaban. Además, este framework resuelve patrones como la inversión de control y la inyección de dependencias [77] El proyecto Spring es inmenso y consta de varios módulos o sub-proyectos. En el caso de las aplicaciones aquí desarrolladas, se optó por utilizar el módulo de Spring Boot [78] ya que facilita la tarea de tener una aplicación lista para ser desplegada en producción. El framework nos proporciona las herramientas para resolver los siguientes puntos:

- Exponer servicios HTTP
- Implementar servicios HTTP respetando las recomendaciones REST, incluyendo HATEOAS
- Fácil puesta-a-punto inicial del proyecto
- Implementar controladores y servicios bajo el esquema de inyección de dependencias
- Genera un archivo JAR auto-contenido, el cual embebe un servidor HTTP (Tomcat [79], Jetty [80])
- Manejo de formatos en las peticiones y respuestas HTTP

Más allá del framework Spring, fueron necesarias otras librerías de software para todo el desarrollo. Principalmente fueron requeridas librerías que Apache Hadoop pone a disposición de los desarrolladores para que podamos interactuar con el ecosistema de Hadoop. En este sentido, existen interfaces que nos permite interactuar con el sistema de archivos (HDFS) como también con el manejador de recursos (YARN), habilitando desde opciones básicas hasta modos avanzados. Es con estas librerías que fue posible desarrollar tareas personalizadas de YARN y servicios que nos permitan manipular archivos almacenados en el HDFS.

Para guardar registro de las actividades que ocurren en las aplicaciones, se eligió la librería SLF4J [81] que permite guardar la bitácora de la aplicación en archivos en el disco rígido, de manera de poder ser inspeccionados en un futuro.

Por último, para el versionado de código se utilizó GIT [82] el popular versionador diseñado por Linus Torvalds. Como plataforma de desarrollo colaborativo se eligió GitHub [83]

Para el manejo de peticiones y respuestas se optó por el formato JSON [84] por su amplia utilización diferentes APIs existentes en el mercado.

6.2 Ejecución y seguimiento de trabajos (Hadoop Manager)

El primer desarrollo realizado para este trabajo fue un administrador de trabajos simplificado, llamado “Hadoop-manager”. Su objetivo es facilitar la administración y ejecución de tareas YARN, habilitando a conocer el estado de las mismas en todo momento.

Sus responsabilidades abarcan desde conocer los tipos de trabajos disponibles a ser ejecutados, permitir ejecutar un trabajo, hacer seguimiento de los diferentes trabajos ejecutados, y hasta eliminar un trabajo del sistema de seguimiento. Estas funcionalidades se exponen a través de servicios HTTP, los cuales pueden ser consumidos con cualquier cliente HTTP, incluso un navegador web.

El recurso principal de esta aplicación es el *trabajo*. Cada trabajo que se ejecute se identificará de manera única con un UUID [85] (Universally Unique Identifier o Identificador único universal). Es con ese identificador que luego podremos consultar su estado o realizar diferentes acciones sobre el trabajo.

El diagrama de las principales clases involucradas y desarrolladas se adjunta en la figura 6.1

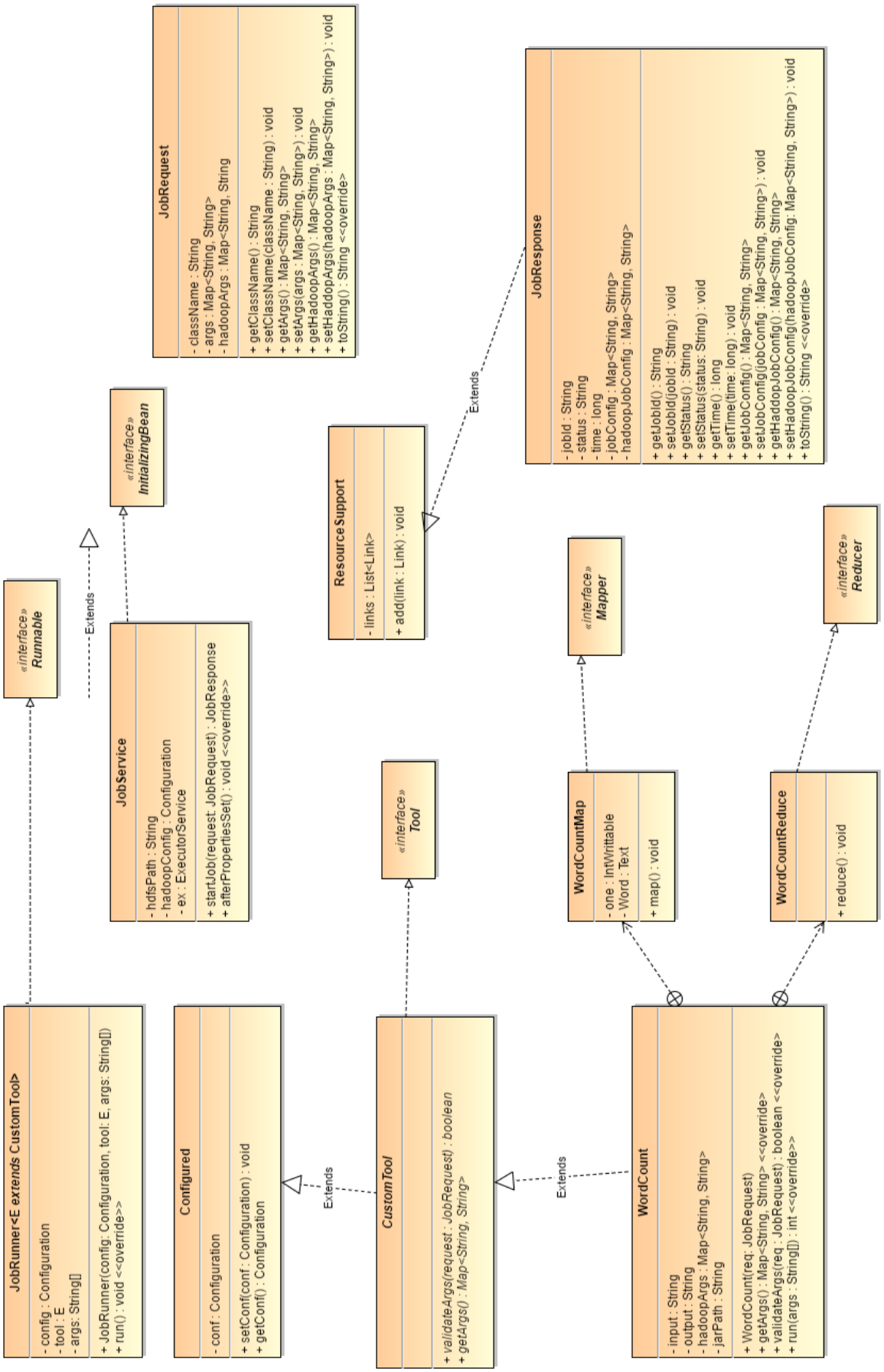


FIGURA 6.1 DIAGRAMA DE CLASES: HADOOP MANAGER

El núcleo de la aplicación es la clase ***JobService*** que se encarga de lanzar trabajos de acuerdo a las peticiones del usuario. El comportamiento de esta funcionalidad se grafica en el siguiente diagrama de secuencia:

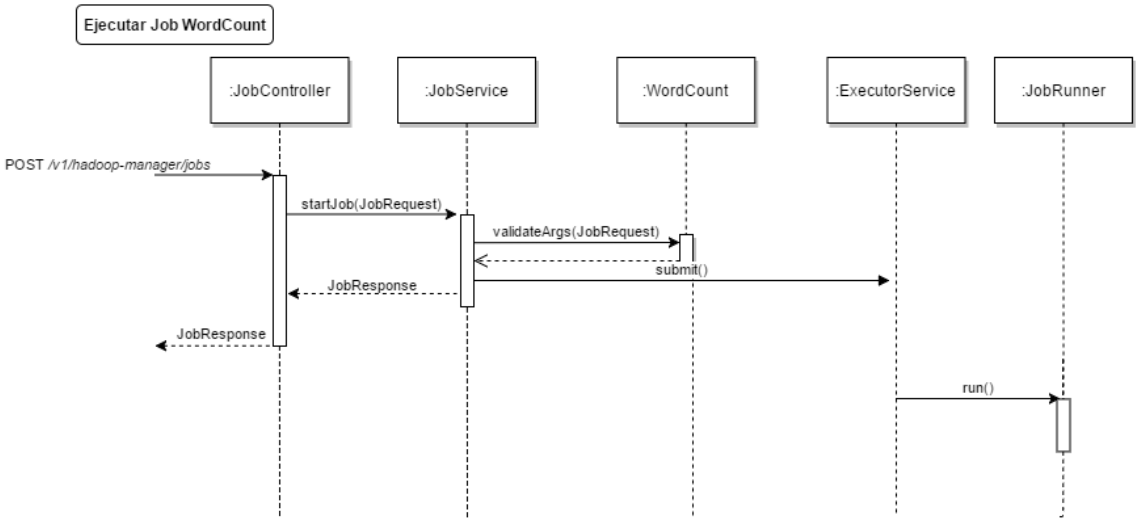


FIGURA 6.2 DIAGRAMA DE SECUENCIA: EJECUTAR JOB WORD COUNT

El servicio se encarga de realizar las validaciones que correspondan al job a ejecutar. Si no existen errores al validar (generalmente son validaciones de argumentos de entrada), se utiliza la clase ***ExecutorService***, perteneciente a la librería Java Concurrent para lanzar la ejecución del Job de manera asincrónica. De esta manera el usuario final no queda a la espera de la finalización del job, sino que podrá, luego, ir conociendo su evolución y estado.

Este ejemplo ilustra de manera clara el funcionamiento de la herramienta desarrollada. El servicio encargado de ejecutar un nuevo Job reúne todas las funcionalidades desarrolladas, por eso fue elegido para explicar en detalle. El mismo espera un llamado HTTP a través del método POST a la URL /v1/hadoop-manager/jobs con el siguiente formato de cuerpo de mensaje:

```
{
  "className": String,
  "args": { Map<String, String> }
}
```

Además de este servicio, la herramienta pone a disposición del usuario las siguientes funcionalidades:

- **GET** */v1/hadoop-manager/jobs/templates*: Retornará un listado con los nombres de las clases Java que representen trabajos disponibles para ser ejecutados. Estos nombres que retorne nos servirán para indicar cuál es el trabajo que nos interesa ejecutar.
- **GET** */v1/hadoop-manager/jobs*: Retornará una lista con el detalle de los trabajos que se encuentren disponibles en el sistema de seguimiento.
- **GET** */v1/hadoop-manager/jobs/**{id}***: Retorna el detalle del trabajo almacenado en el sistema de seguimiento a partir de la identificación {id}. Entre la información devuelta se incluye: la identificación del trabajo, su estado actual, el tiempo consumido en ejecución, y los parámetros con los que fue ejecutado

- **DELETE** */v1/hadoop-manager/jobs/**{id}***: Elimina al trabajo del sistema de seguimiento. Es ideal para casos donde terminó el trabajo, y ya hemos consumido toda la información que necesitábamos.

6.3 Cliente HDFS (HDFS-REST)

Como parte de las pruebas que fueron realizadas para este trabajo, se incluye la interacción de servicios externos con el sistema de archivos de Apache Hadoop. Es por esto que se desarrolló una herramienta para facilitar la interacción entre el usuario y el sistema de archivos de Hadoop, HDFS. Este desarrollo simplifica las operaciones más populares que incluyen: crear, borrar, obtener un archivo y/o carpeta. Para todas las operaciones es necesario indicar la ruta sobre la que vamos a trabajar, indicada siempre en el parámetro "*path*". En esta aplicación, los recursos principales son el archivo (*file*) y la carpeta (*folder*).

Para conocer las principales clases que componen a esta herramienta, se adjunta el diagrama de clases correspondiente en la figura 6.3:

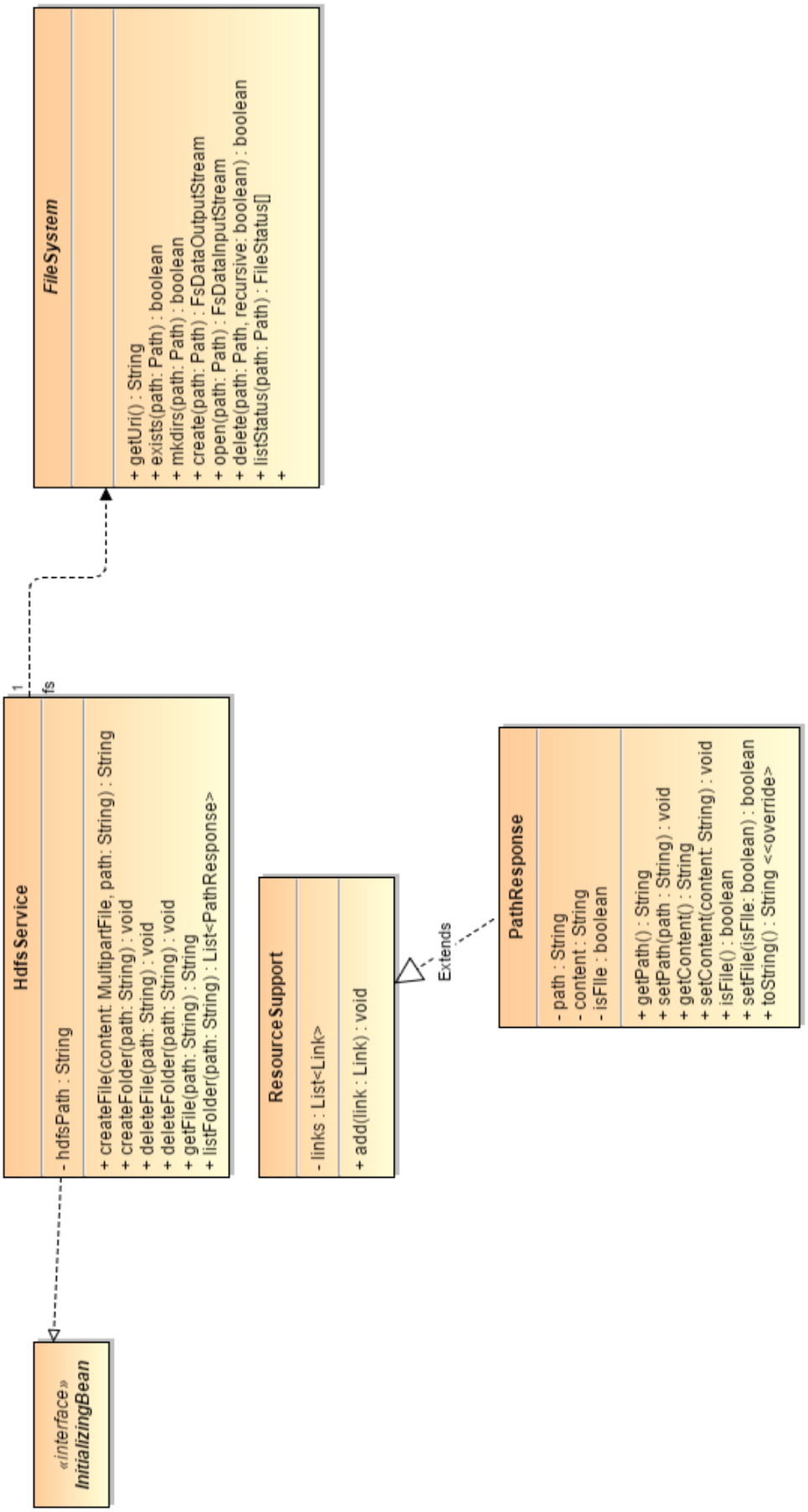


FIGURA 6.3 DIAGRAMA DE CLASES: HDFS REST

La forma de interactuar con la herramienta es muy similar entre los servicios que expone. Por este motivo se muestra en la figura 6.4 el diagrama de secuencia para el servicio de creación de archivos. En este servicio intervienen todas las características desarrolladas:

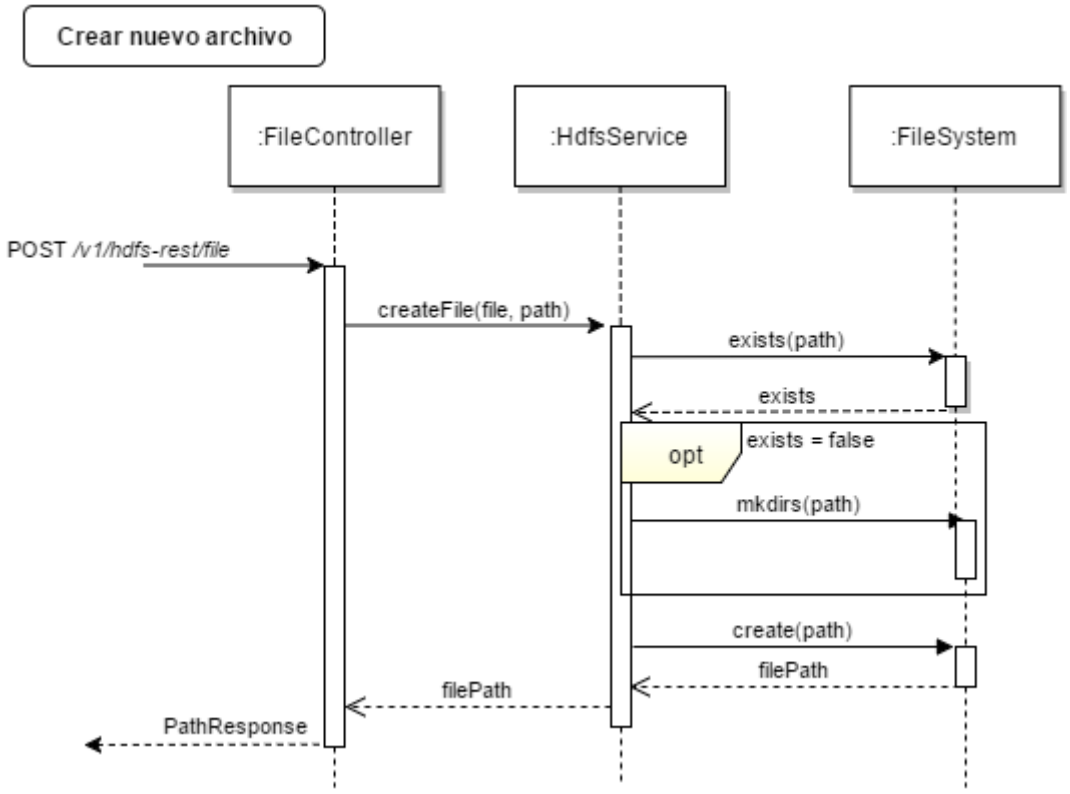


FIGURA 6.4 DIAGRAMA DE SECUENCIA: CREAR ARCHIVO EN HDFS

La creación de un archivo comienza cuando el cliente (por ejemplo el navegador web) realiza una llamada HTTP **POST** a la ruta */v1/hdfs-rest/file* indicando la ruta donde se va a alojar el archivo y enviando el contenido del mismo. Lo que sigue luego, es responsabilidad del principal servicio de esta herramienta, **HdfsService**. Éste se encarga de realizar las validaciones correspondientes, como saber si la carpeta sobre la cual se va a alojar el archivo existe, o debe ser creada. Si es necesario, se crea en ese momento, para luego pasar a la creación del archivo propiamente dicha, que consiste de una operación de escritura en el HDFS. La respuesta final será la ruta de acceso completa hacia el archivo guardado en el sistema de archivos de Hadoop, junto con la información que ofrece HATEOAS para navegar el recurso.

Además de este servicio, nuestra herramienta nos ofrece otras operaciones que se exponen bajo las siguientes URL:

- **GET** */v1/hdfs-rest/file?path=**{path}*** : Este servicio retorna el contenido del archivo alojado en el HDFS bajo la ruta indicada en el parámetro path. En el caso de que dicha ruta no exista en el HDFS, se retornará un error.
- **DELETE** */v1/hdfs-rest/file?path=**{path}***: El servicio se encarga de eliminar el archivo ubicado en el sistema de archivos de Hadoop, localizado en la ruta que se describa en el parámetro path. La respuesta no tiene cuerpo de mensaje. El éxito o no de la operación, se determina a partir del estado Http

Además se brindan servicios muy similares, pero que operan sobre las carpetas del sistema de archivos:

- GET /v1/hdfs-rest/folder?path={path}: Retornará un listado de archivos y carpetas que se encuentren bajo la ruta indicada en el parámetro path, junto con las operaciones permitidas en cada archivo o carpeta
- DELETE /v1/hdfs-rest/folder?path={path}: Realiza un borrado recursivo del directorio indicado en el parámetro path.
- POST /v1/hdfs-rest/folder?path={path}: Este servicio crea una carpeta en la ruta indicada en el parámetro path (ruta absoluta). Retorna información de la carpeta creada, junto con las operaciones que permite.

Con los servicios web que exponen estas aplicaciones, estamos en condiciones de realizar los diferentes experimentos. Es importante remarcar que ambas APIs cumplen la mayoría de las recomendaciones REST, lo que hace que sean interfaces portables y extensibles. Ambas aplicaciones hacen utilización de los estados y verbos HTTP para indicar operaciones sobre recursos y estados de respuestas. También hacen uso de HATEOAS para lograr una fácil navegación, y se identifican a los recursos de manera unívoca.

6.4 Antecedentes

Previo a exponer los experimentos realizados, será de gran utilidad conocer las aplicaciones similares que existen hoy en el mercado. De esta manera entenderemos las ventajas y desventajas de cada una. Se nombrarán antecedentes similares tanto del administrador de trabajos de YARN, como también de la interfaz de interacción con el sistema de archivos de Hadoop.

6.4.1 Antecedentes de Hadoop Manager

Apache Eagle

Apache Eagle [86] es una herramienta de código abierto que permite identificar problemas relacionados a la seguridad y a la performance en plataformas de Big Data. Es decir, no sólo funciona sobre Hadoop, sino también en otros ambientes como ser Apache Spark o alguna tecnología noSQL. La comunidad de Apache ha decidido que el proyecto Eagle forme parte del conjunto “Apache Incubator”, lo que claramente lo identifica con un potencial de crecimiento y aceptación por parte de la comunidad de desarrolladores.

La herramienta funciona analizando datos de aplicaciones YARN, junto con métricas que se expongan por JMX, logs de aplicación, entre otras fuentes de datos. Cuenta además, con un motor de alertas para identificar y tratar errores de seguridad o performance. Es por esto que su arquitectura presenta un grado de complejidad avanzado:

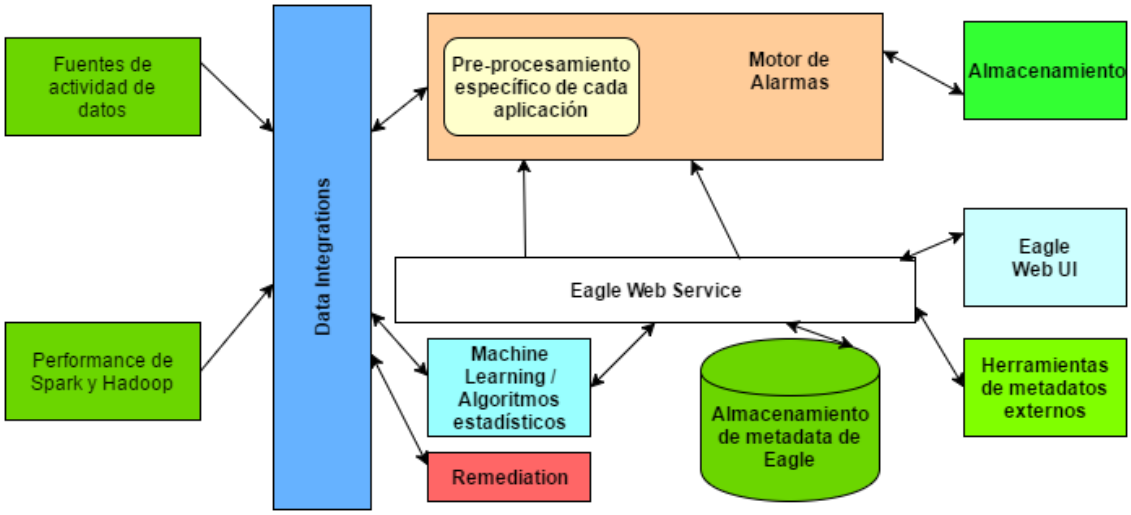


FIGURA 6.5 DIAGRAMA DE ARQUITECTURA DE APACHE EAGLE

Como se muestra en la figura 6.5, son varias las componentes que integran a esta herramienta. Las "fuentes de actividad de datos", como los logs de aplicación, y la "información de performance de Spark y Hadoop", como pueden ser las métricas expuestas por JMX o las métricas de las ejecuciones de MapReduce, forman juntas lo que son las entradas de información a la herramienta. El núcleo de su arquitectura es el motor de alarmas en tiempo real, el cual es altamente escalable. Incluye un coordinador, un servicio de almacenamiento y de metadatos. Las demás componentes cumplen diferentes funciones, desde visualizar la información a través del módulo Eagle Web UI, como también almacenar datos.

Una característica importante es que la comunidad puede integrar más fuentes de datos y se pueden escribir aplicaciones que estén a la espera de una alerta para tomar decisiones (por ejemplo, la automatización de un procedimiento para solucionar determinados escenarios)

La herramienta es muy poderosa y debe considerarse su instalación en ambientes de Big Data corporativos, donde la seguridad de los datos es crítica para el negocio. Sin dudas agrega muchas más funcionalidades a la herramienta propuesta para este trabajo, por lo que agregaría un nivel de dificultad no deseado para las pruebas que se realizarán.

Azkaban

Azkaban [87], desarrollado por el equipo de Linkedin, es un planificador de flujo de trabajos de Hadoop que se ejecutan en lote. Apunta principalmente a planificar las dependencias entre trabajo, y también ofrece una interfaz de usuario para conocer el estado del flujo de trabajo.

Posee tres componentes claves, ilustradas en la figura 6.6, lo que simplifica la tarea de conocer y comprender su arquitectura:

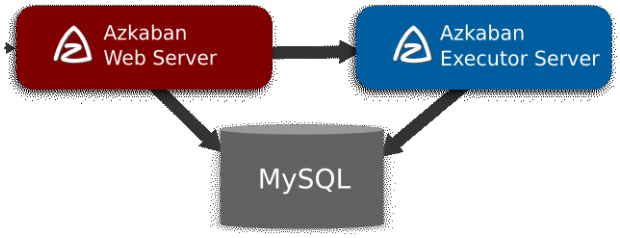


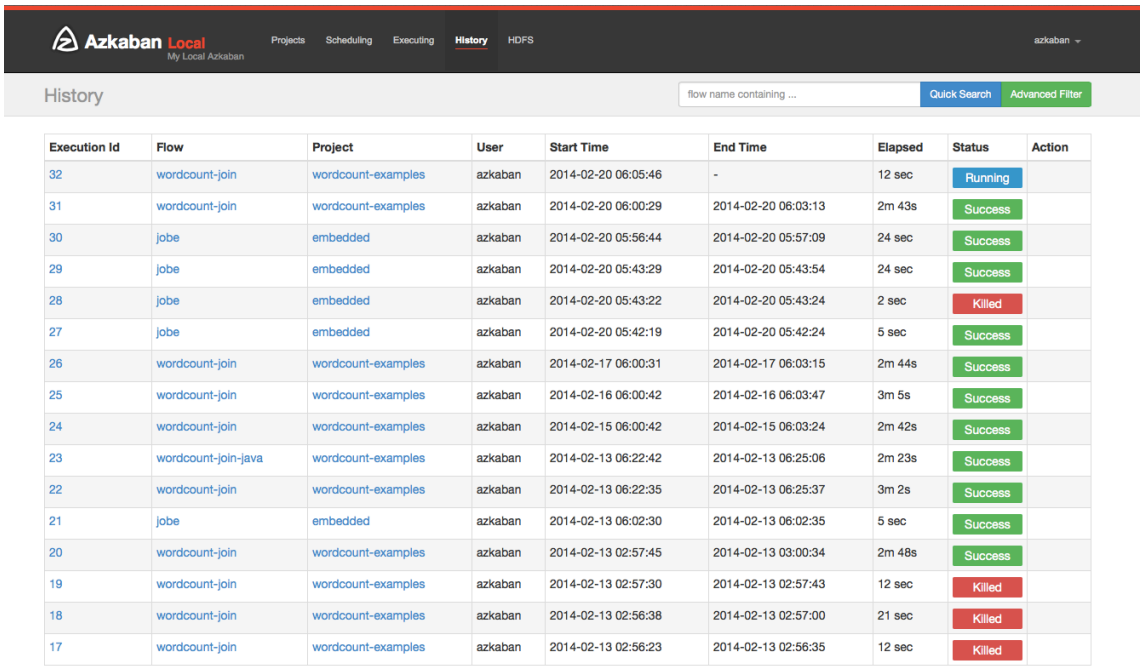
FIGURA 6.6 COMPONENTES DE AZKABAN

Azkaban usa la base de datos MySql para guardar mucha información sobre su estado. Entre otros datos, almacena: los proyectos creados y sus permisos, el estado de los flujos de ejecución, información sobre la planificación de tareas, logs, dependencias entre flujos, entre otra información.

El servidor web de Azkaban es la componente principal. Se encarga de la administración de proyectos, autenticación, planificación y monitoreo de las ejecuciones. Además es quien sirve la interfaz web al usuario final.

El servidor de ejecución se encarga justamente de lanzar los diferentes flujos de trabajo, haciendo el seguimiento y guardando los diferentes eventos y estados que puedan ocurrir durante la ejecución.

La interfaz web, plasmada en la figura 6.7 muestra al usuario información similar a la que mostramos con la implementación de nuestra herramienta para la ejecución y monitoreo de tareas en YARN: Tenemos columnas que identifican a la información: una identificación única (id) de la ejecución, el nombre que se le dio al flujo, el proyecto al cual pertenece. Además nos ofrece la hora de comienzo y fin de flujo, junto con el tiempo transcurrido, y el estado del mismo, distinguido principalmente en 3 colores: azul, indicando que el flujo está en ejecución; verde, para indicar que el flujo terminó exitosamente; rojo, para indicar que el flujo terminó con error. Por último nos ofrece una columna con acciones a tomar sobre ese flujo: depende el estado del flujo, mostrará o no ciertas acciones.



Execution Id	Flow	Project	User	Start Time	End Time	Elapsed	Status	Action
32	wordcount-join	wordcount-examples	azkaban	2014-02-20 06:05:46	-	12 sec	Running	
31	wordcount-join	wordcount-examples	azkaban	2014-02-20 06:00:29	2014-02-20 06:03:13	2m 43s	Success	
30	jobe	embedded	azkaban	2014-02-20 05:56:44	2014-02-20 05:57:09	24 sec	Success	
29	jobe	embedded	azkaban	2014-02-20 05:43:29	2014-02-20 05:43:54	24 sec	Success	
28	jobe	embedded	azkaban	2014-02-20 05:43:22	2014-02-20 05:43:24	2 sec	Killed	
27	jobe	embedded	azkaban	2014-02-20 05:42:19	2014-02-20 05:42:24	5 sec	Success	
26	wordcount-join	wordcount-examples	azkaban	2014-02-17 06:00:31	2014-02-17 06:03:15	2m 44s	Success	
25	wordcount-join	wordcount-examples	azkaban	2014-02-16 06:00:42	2014-02-16 06:03:47	3m 5s	Success	
24	wordcount-join	wordcount-examples	azkaban	2014-02-15 06:00:42	2014-02-15 06:03:24	2m 42s	Success	
23	wordcount-join-java	wordcount-examples	azkaban	2014-02-13 06:22:42	2014-02-13 06:25:06	2m 23s	Success	
22	wordcount-join	wordcount-examples	azkaban	2014-02-13 06:22:35	2014-02-13 06:25:37	3m 2s	Success	
21	jobe	embedded	azkaban	2014-02-13 06:02:30	2014-02-13 06:02:35	5 sec	Success	
20	wordcount-join	wordcount-examples	azkaban	2014-02-13 02:57:45	2014-02-13 03:00:34	2m 48s	Success	
19	wordcount-join	wordcount-examples	azkaban	2014-02-13 02:57:30	2014-02-13 02:57:43	12 sec	Killed	
18	wordcount-join	wordcount-examples	azkaban	2014-02-13 02:56:38	2014-02-13 02:57:00	21 sec	Killed	
17	wordcount-join	wordcount-examples	azkaban	2014-02-13 02:56:23	2014-02-13 02:56:35	12 sec	Killed	

FIGURA 6.7 HISTORIA DE EJECUCIÓN EN AZKABAN

La herramienta es muy recomendada cuando la ejecución de trabajos de Hadoop presenta dependencias. Para el caso de ejecuciones simples, implicaría agregar una componente más a la arquitectura, de lo cual no se sacarían mayores ventajas.

6.4.2 Antecedentes de HDFS Rest

WebHDFS

WebHDFS [88] es una API HTTP que provee Hadoop para interactuar con el sistema de archivos HDFS. Soporta todas las operaciones que ofrece HDFS, agregando la opción de un nivel de seguridad con autenticación.

El utilitario viene disponible con la instalación de Hadoop, y sólo es necesario agregar una propiedad en el archivo *hdfs-default.xml* para que se habilite su funcionamiento.

Todas las operaciones que fueron implementadas en “HDFS Rest” se encuentran disponibles en WebHDFS. Sin embargo, nuestro desarrollo respeta mejor las recomendaciones sobre una implementación REST. Además, WebHDFS soporta muchas operaciones, lo que le agrega un nivel de complejidad en su implementación.

Por nombrar un caso puntal, la creación de un archivo mediante WebHDFS involucra dos pasos, es decir, dos llamadas HTTP. En la primera indicamos cual es el archivo que queremos crear. Esto se hace al NameNode donde atiende el servidor WebHDFS. La respuesta de esta petición nos dirá en cuál DataNode debemos crear el archivo. Con esta información, haremos el segundo llamado HTTP, enviando el contenido del archivo.

Lo mismo sucede para obtener un archivo: primero debe consultarse de cuál DataNode leer, y luego acceder al mismo para obtener el contenido del archivo.

En este sentido, nuestra herramienta desarrollada nos genera una capa de abstracción, ya que no nos interesa el nodo donde está almacenado el archivo, sino que accedemos por su ruta que es única a nivel clúster.

HttpFS

HttpFS [89] es una herramienta similar a WebHDFS, a tal punto que son intercambiables casi sin costo alguno. En este sentido, es un servidor que provee acceso al sistema de archivos HDFS a través de una API HTTP.

La principal diferencia con WebHDFS es que HttpsFs corre como un servicio totalmente separado del núcleo de Hadoop. Es decir, HttpFs es una aplicación web Java que se ejecuta bajo su propio Tomcat pre-configurado.

Las principales ventajas de utilizar HttpFs incluyen:

- La posibilidad de transferir datos entre clústeres que corren diferentes versiones de Hadoop. De esta manera se evitan los problemas que puedan surgir por el versionado que ofrece RPC.
- Puede ser utilizado como único punto de acceso controlado para un clúster que se ejecuta detrás de un firewall.
- Se puede acceder a los datos con cualquier utilitario HTTP como ser cURL o wget.
- Permite la implementación propia de una interfaz de usuario que consuma los servicios HTTP.

Si bien la herramienta es muy cómoda, aún no cumple con muchas de las recomendaciones REST. Además sigue agregando complejidad tanto para las peticiones como también para leer las respuestas que brinda su API.

Estas herramientas presentadas son un claro antecedente de los desarrollos presentados en esta tesina. De ellas se rescatan sus principales ventajas y funcionalidades, adaptando algunas de ellas para el objetivo de este trabajo. No es la intención presentar un reemplazo de alguna de ellas, sino la de tener un utilitario que se adapte a la necesidad de este trabajo, y que nos permita hacer el foco en nuestro principal objetivo: mostrar la ejecución de un servicio en un ambiente clusterizado a través de una interfaz REST.

Capítulo 7 - PRUEBAS REALIZADAS

En este capítulo utilizamos las herramientas desarrolladas con los siguientes fines:

- Conocer el costo que genera la transferencia de archivos desde el sistema de archivos del sistema operativo hacia el sistema de archivos de Apache Hadoop, HDFS
- Demostrar la posibilidad de ejecución de un trabajo en YARN en un ambiente de computación en la nube, a través de la interfaz REST que se desarrolló
- Alimentar a un trabajo de YARN desde múltiples fuentes de datos, siendo esto posible gracias a la interfaz REST que interactúa con el sistema de archivos.

El trabajo que se decidió ejecutar para estas pruebas es un trabajo de MapReduce llamado "Word Count", o "contador de palabras". El trabajo podría ser otro ejemplo, ya que va más allá de lo que se quiere plasmar en el ejemplo. El punto aquí es mostrar la forma de interactuar a través de la interfaz HTTP REST, además de trabajar en los objetivos previamente mencionados.

Lo que hace este trabajo es, dada una entrada, contar las ocurrencias de cada palabra de dicha entrada, y plasmarlo en un archivo de salida.

Como todo trabajo de MapReduce, consta de dos etapas:

- Map: Analiza el archivo de entrada, y por cada palabra produce un par de salida *(palabra, 1)*
- Reduce: Recibe un listado de la salida del map, todos con la misma palabra. Luego cuenta los valores y produce una salida del tipo *(palabra, n)*, donde "n" es el número de ocurrencias de la palabra.

Ejemplo de ejecución:

```
Entrada: "hola hola a todos todos"
```

Produce un archivo de salida con el siguiente formato:

```
hola  2
a     1
todos 2
```

Este es un ejemplo sencillo pero al mismo tiempo poderoso, ya que reúne toda la funcionalidad que necesitamos para nuestro experimento. Su implementación es simple, lo que nos ahorra el

trabajo de entender el código y nos permite focalizarnos de lleno en el experimento y en lo que se ha desarrollado para el acceso a través de una interfaz REST a la infraestructura de cómputo.

7.1 Ambiente de pruebas

El experimento se realiza sobre un clúster de 3 computadoras del perfil “Básico A – A2” incluido dentro de Microsoft Azure. Este perfil posee 2 núcleos de procesamiento, 3.50 gb de memoria RAM y capacidad de almacenamiento (disco rígido) de 60 gb.

Es necesario que en la máquina que usemos como maestra instalemos la herramienta GIT para obtener el código fuente del proyecto, y también Maven para empaquetar el mismo y poder ejecutarlo.

```
# Instalación de GIT
$ sudo apt-get install git

# Instalación de Maven
$ sudo apt-get install maven
```

Instalación de Hadoop Manager y HDFS-Rest

Para dejar disponibles al uso las dos herramientas desarrolladas debemos realizar la operación de clonado de ambos repositorios de código, y por último empacar los proyectos para que se generen los archivos “JAR”

```
# Instalación de Hadoop Manager

$ cd $HOME
$ git clone https://github.com/albarra/hadoop-manager.git
$ cd hadoop-manager/ && mvn package

# Instalación de HDFS Rest

$ cd $HOME
$ git clone https://github.com/albarra/hdfs-rest.git
$ cd hdfs-rest/ && mvn package
```

Además, debemos asegurarnos de que la máquina que actúe como maestra tenga habilitados los puertos por los cuales se acceden a los recursos de “Hadoop Manager” (8081) y “HDFS Rest” (8080), tal como se muestra en la figura 7.1.

FIGURA 7.1 CONFIGURACIÓN DE EXTREMOS PARA LAS APLICACIONES DESARROLLADAS

7.2 Experimento - Transferencia de archivos desde múltiples fuentes de datos

El primer experimento práctico consta de transferir archivos desde cualquier fuente hacia el sistema de archivos de Hadoop, HDFS. Se comparará luego su comportamiento con la transferencia nativa que provee Hadoop para mover archivos desde nuestro sistema de archivos hacia HDFS.

En este sentido debemos tomar dos muestras: Una será utilizando la API que ofrece nuestro desarrollo, HDFS Rest, para crear archivos en el sistema de archivos HDFS. La otra muestra se tomará utilizando la herramienta nativa de Hadoop, "hdfs", que nos permite interactuar con el sistema de archivos distribuido.

Se plantean escenarios de prueba con archivos de 1, 10, 100, y 500mb

Previamente debemos asegurarnos de tener los servicios de Hadoop funcionando, y además debemos ejecutar la aplicación HDFS Rest

```
# Sobre la carpeta "target" que se genera en nuestra
aplicación, ejecutar el JAR generado

$ java -Xms512m -Xmx2g -jar hdfs-rest-0.0.2-SNAPSHOT.jar
```

En este caso fue necesario agregar algunas opciones a la Java Virtual Machine (JVM) para su óptimo funcionamiento [90]

Transferencia directa:

Para la transferencia directa de archivos, utilizamos el utilitario nativo de Hadoop, con el cual también crearemos las carpetas que se usarán como destino.

```
$ hdfs dfs -mkdir -p /upload_direct/1mb
$ hdfs dfs -mkdir -p /upload_direct/10mb
$ hdfs dfs -mkdir -p /upload_direct/100mb
$ hdfs dfs -mkdir -p /upload_direct/500mb
```

Una vez creadas las carpetas, se proceden a ejecutar las transferencias, con el comando "time" [91] que nos facilitará la medición de tiempos

```
$ time hdfs dfs -copyFromLocal 1mb /upload_direct/1mb
$ time hdfs dfs -copyFromLocal 10mb /upload_direct/10mb
$ time hdfs dfs -copyFromLocal 100mb /upload_direct/100mb
$ time hdfs dfs -copyFromLocal 500mb /upload_direct/500mb
```

Transferencia a través de la herramienta desarrollada

La transferencia de archivos, haciendo uso de la API REST desarrollada, se realiza a través del comando cURL [92]. En los *logs* de la aplicación podremos observar el tiempo demandado para la operación de transferir un archivo hacia el HDFS. Para las pruebas realizadas, se descarta el tiempo consumido de transferencia. En este caso, no es necesaria la previa existencia de las carpetas destino:

```
$ curl -i -F file=@1mb http://localhost:8080/v1/hdfs-
rest/file?path=/upload_rest/1mb/ --verbose

$ curl -i -F file=@10mb http://localhost:8080/v1/hdfs-
rest/file?path=/upload_rest/10mb/ --verbose

$ curl -i -F file=@100mb http://localhost:8080/v1/hdfs-
rest/file?path=/upload_rest/100mb/ --verbose

$ curl -i -F file=@500mb http://localhost:8080/v1/hdfs-
rest/file?path=/upload_rest/500mb/ --verbose
```

Los tiempos consumidos en todas las operaciones se ilustran en la siguiente tabla:

TABLA 7-1 TIEMPOS DE TRANSFERENCIA: HDFS REST VS HDFS NATIVO

Tiempo / Tamaño	HDFS REST	Hadoop HDFS Nativo
1mb	1.40 seg	6.76 seg
10mb	1.04 seg	7.12 seg
100mb	6.06 seg	11.22 seg
500mb	29.34 seg	34.61 seg

Es importante remarcar que los dos tipos de transferencias fueron probados en igualdad de condiciones, bajo el mismo Hardware, y con los mismos archivos de origen.

A simple vista se puede observar una clara ventaja en velocidad de operación a favor de la herramienta desarrollada. Además, nuestra herramienta desarrollada no requiere la previa creación de la carpeta destino, y el comando necesario a ejecutar, cURL, es ampliamente conocido.

Otra ventaja fundamental de la herramienta desarrollada es la abstracción en cuanto a la tecnología. El usuario final no tiene necesidad de conocer siquiera la existencia de Hadoop. Sólo necesita transferir archivos, y para esto, lo único que debe conocer es la API que provee la herramienta. Con esto dicho, la curva de aprendizaje de uso de la herramienta desarrollada es mucho menos costosa que la curva que puede presentar el uso de Hadoop, sobre todo si se va a trabajar con usuarios que no son experimentados.

7.3 Experimento – Ejecución de trabajo en un ambiente en la nube a través de una interfaz REST

El segundo experimento es fundamental en el desarrollo de la presente tesina. Se ejecutará un trabajo de MapReduce en el clúster constituido, a través de la herramienta desarrollada para este fin, Hadoop Manager. La ejecución de este trabajo se hará de manera remota, haciendo uso de la interfaz http que expone la herramienta, la cual se considera una interfaz REST.

Es importante remarcar que la prueba ejecuta un trabajo de Map Reduce, pero el desarrollo propuesto es lo suficientemente flexible para ejecutar cualquier tarea de YARN en el ecosistema de Hadoop.

El trabajo a ejecutar, llamado WordCount, utilizará como parámetros de entrada los archivos que fueron previamente depositados en el HDFS utilizando la interfaz REST desarrollada en este informe. Se trabaja con archivos de 1, 10 y 100mb. En este sentido, el trabajo recibe la entrada

desde múltiples fuentes de datos. Cualquier dispositivo con conexión a internet podrá aportar archivos de entrada al trabajo de WordCount, desde computadoras de escritorio, portátiles, teléfonos inteligentes, tablets, entre otros.

Se muestran a continuación las ejecuciones lanzadas. Recordar que previamente debemos asegurarnos de tener los servicios de Hadoop funcionando, y además debemos ejecutar la aplicación Hadoop Manager

```
# Sobre la carpeta "target" que se genera en nuestra
aplicación, ejecutar el JAR generado

$ java -Xms512m -Xmx2g -jar hadoop-manager-0.0.1-
SNAPSHOT.jar
```

Ejecución con tamaño de entrada de 1mb: Tiempo de ejecución 45478 ms

```
$curl -H "Content-Type: application/json" -X POST -d
'{"className": "com.unlp.my.jobs.WordCount", "args":
{"input": "/upload_rest/1mb", "output": "/output1mb", "jar":
"/home/jcc2015/hadoop-manager/target/hadoop-manager-0.0.1-
SNAPSHOT.jar"}}' http://localhost:8081/v1/hadoop-
manager/jobs
```

Ejecución con tamaño de entrada de 10mb: Tiempo de ejecución 57004 ms

```
$ curl -H "Content-Type: application/json" -X POST -d
'{"className": "com.unlp.my.jobs.WordCount", "args":
{"input": "/upload_rest/10mb", "output": "/output10mb",
"jar": "/home/jcc2015/hadoop-manager/target/hadoop-manager-
0.0.1-SNAPSHOT.jar"}}' http://localhost:8081/v1/hadoop-
manager/jobs
```

Ejecución con tamaño de entrada de 100mb: Tiempo de ejecución 198380 ms

```
$ curl -H "Content-Type: application/json" -X POST -d
'{"className": "com.unlp.my.jobs.WordCount", "args":
{"input": "/upload_rest/100mb", "output": "/output100mb",
"jar": "/home/jcc2015/hadoop-manager/target/hadoop-manager-
0.0.1-SNAPSHOT.jar"}}' http://localhost:8081/v1/hadoop-
manager/jobs
```

La ejecución de trabajos bajo esta modalidad no requiere conocimientos sobre Hadoop, aunque se necesita conocer el trabajo que se desea ejecutar, junto con sus parámetros soportados. Es decir, al igual que la API de HDFS Rest, permite una abstracción sobre la tecnología que estemos utilizando.

Al ser una API navegable, luego de lanzar la ejecución podremos conocer el estado del trabajo, haciendo seguimiento del mismo hasta que finalice. Esto lo vemos en estado del trabajo, como se muestra a continuación:

```
$ curl -I http://jcc2015.cloudapp.net:8081/v1/hadoop-
manager/jobs/07e7b033-d2bc-40b7-9162-cdf1d2510134

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Sun, 09 Oct 2016 19:50:54 GMT


{
  "jobId": "07e7b033-d2bc-40b7-9162-cdf1d2510134",
  "status": "SUCCEEDED",
  "time": 45478,
  "jobConfig": {
    "input": "/upload_rest/1mb",
    "output": "/output1mb-a",
    "jar": ...

...
```

Este experimento muestra la evolución del desarrollo presentado en las III Jornadas de Cloud Computing del año 2015 [93], titulado “Hadoop en Cloud: Envío y Procesamiento Asincrónico de Datos”. En esa ocasión se proponía la ejecución asincrónica del trabajo, al igual que en este experimento. Sin embargo las múltiples fuentes de entrada que deseaban participar del trabajo, debían facilitar sus archivos a través de la interfaz SCP [94]. Luego, a través de una señal, se lanzaba la ejecución del trabajo. En el ejemplo presentado en las Jornadas, la señal consistía en subir un archivo llamado “FINISH”, sin contenido. Con un script, detectábamos esta señal, para luego lanzar la ejecución del trabajo directamente, con parámetros ya definidos.

Este experimento muestra la evolución planteada a partir de dicha presentación en las Jornadas, pues ahora el lanzamiento de un nuevo trabajo se logra exclusivamente a través de una interfaz REST. Además, las fuentes de entrada también se proporcionan por una interfaz REST de sencilla utilización. Se logró además, mayor flexibilidad en los parámetros que acepta la ejecución de un trabajo. Por último, gracias al uso de clases, subclases, e interfaces, fue posible desarrollar una herramienta flexible y extensible a nuevos trabajos, y no sólo el presentado en este ejemplo.

Capítulo 8 - CONCLUSIONES Y TRABAJO FUTURO

Desde el momento en que se definió la propuesta de la presente tesis, se ha recorrido un camino en el que se vieron involucradas tareas de investigación y desarrollo en conjunto. El objetivo central se basó en demostrar la posibilidad de utilizar servicios en ambientes clusterizados, administrados bajo interfaces REST, haciendo además que estos servicios puedan ser alimentados desde múltiples fuentes de datos de entrada. Fue por esto que se desarrollaron las dos piezas de software, que sirvieron para la demostración práctica de nuestra propuesta. La prueba experimental fue entonces la ejecución de estos desarrollos propios, haciendo llamados a servicios que se encontraban en ejecución, en la nube. Como hemos visto, estos servicios responden en tiempo y forma a las peticiones realizadas. Por lo dicho anteriormente, se ofrecen las siguientes conclusiones t líneas de trabajo futuro.

8.1 Conclusiones

El desarrollo de la presente tesina ha cumplido todos los objetivos presentados en su propuesta. Desde su comienzo, con la participación en la Jornadas de Cloud Computing & Big Data [93] organizadas por la Facultad de Informática, se ha madurado la idea y la necesidad de demostrar la posibilidad de ejecutar con facilidad servicios en la nube bajo alguna interfaz conocida, como lo es HTTP + REST. También se ha dejado en claro la necesidad que plantean los sistemas actuales en cuanto a soportar muchas fuentes de datos de entrada.

El marco teórico presentado en esta tesina nos invita a conocer el inmenso mundo de la computación en la nube, junto con sus tecnologías más importantes como los son big data y map reduce. Asimismo, conocer una herramienta que permita llevar a la práctica estos conocimientos teóricos es de vital importancia para cualquier persona que quiera hacer aportes a la comunidad dentro de este ámbito. Es por eso que se dedicó un capítulo a Apache Hadoop.

Los desarrollos que se ofrecen (HDFS Rest y Hadoop Manager) llevan a la práctica el objetivo de la tesina, plasmado en su propuesta. Cumplen el rol de demostrar que las ideas propuestas se pueden llevar a cabo en la realidad.

Los desarrollos prácticos son, además, un aporte a la comunidad de desarrollo. Ambos proyectos se encuentran disponibles para cualquier persona que desee hacer aportes o extender sus funcionalidades.

Por último, y no por eso menos importante, los desarrollos aportan un nivel de abstracción respecto de Hadoop. No es necesario conocer en profundidad el funcionamiento de todo el ecosistema de Hadoop. Sólo alcanza conocer su fundamento básico y la manera en que operan las herramientas desarrolladas.

8.2 Trabajo Futuro

La tesina desarrollada propone algunas líneas de trabajo futuro, como ser:

- Realizar un análisis exhaustivo respecto de la performance de las herramientas desarrolladas, utilizando métricas como speed-up, eficiencia, escalabilidad, entre otras
- Generar una herramienta que logre la automatización del proceso de instalación y configuración de Apache Hadoop dentro de un clúster. Las acciones requeridas se pueden llevar a cabo mediante la combinación de diferentes scripts junto con algunos archivos específicos de configuración.
- Estandarizar el proceso de empaquetado y despliegue de las aplicaciones desarrolladas para facilitar su ejecución en ambientes productivos.
- Documentar los desarrollos realizados a través de alguna herramienta para representar APIs que son RESTful, como por ejemplo, Swagger [95]
- Extender “Hadoop Manager” con más ejemplos integrados como es el caso del contador de palabras

APÉNDICE - CREACIÓN DE CLÚSTER HADOOP EN AZURE

En el presente anexo se describen los pasos de configuración necesarios para crear y ejecutar un clúster de Hadoop en el servicio de Microsoft Azure.
Se asume que el usuario ya cuenta con una suscripción de Microsoft Azure, la cual es requerida para consumir los servicios que se requieren en esta guía.

El primer paso es crear un "***Servicio en la nube***", como se muestra en la figura 9.1. Este servicio nos permitirá agrupar un conjunto de máquinas bajo una misma red, y tratarlas como un clúster. También nos permitirá acceder bajo un nombre de dominio: Para crearlo, vamos a "Servicios en la nube", y accedemos a la creación personalizada, donde elegimos un nombre y una región:

FIGURA 9.1 AZURE - CREACIÓN DE SERVICIO EN LA NUBE

Lo que sigue a este paso es crear una red virtual asociada al servicio recién creado (***jcc2015***)
Para esto vamos a ***Redes***, Nuevo, Personalizada. Elegimos un nombre y una región (figura 9.2):

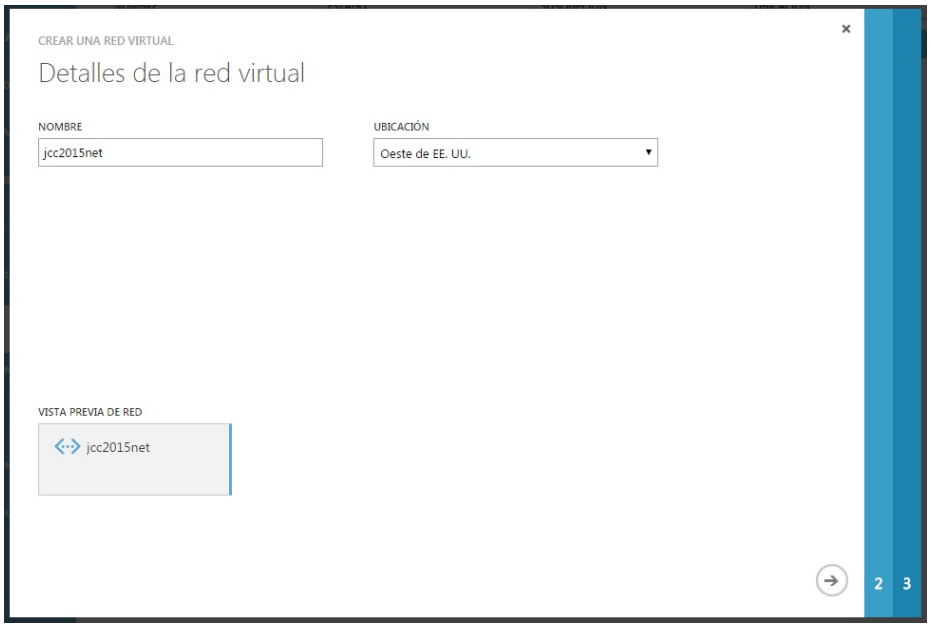


FIGURA 9.2 AZURE - CREACIÓN DE RED VIRTUAL

Damos "siguiente". Las opciones que se requieren en el paso (2) las dejaremos en blanco, y Windows Azure se encargará de asignarlas. En el paso 3 tenemos que configurar el espacio de direcciones: La red, y la subred, la cual en nuestro caso será una sola:

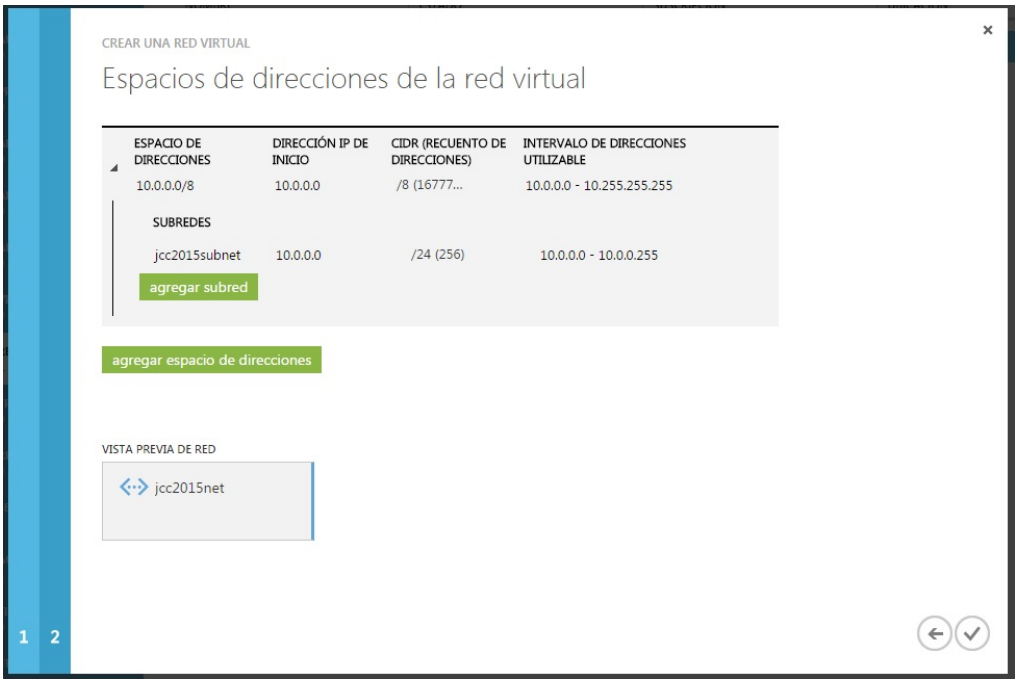


FIGURA 9.3 AZURE - CONFIGURACIÓN DE ESPACIOS DE DIRECCIONES DE LA RED VIRTUAL

Luego debemos crear la máquina virtual que utilizaremos como plantilla para crear todas las máquinas del clúster. Para esto vamos a "***Máquinas virtuales***", luego a "Nuevo" y luego "De la galería". Una vez ahí, seleccionamos *Ubuntu Server 12.04 LTS*, tal como vemos en la figura 9.4.

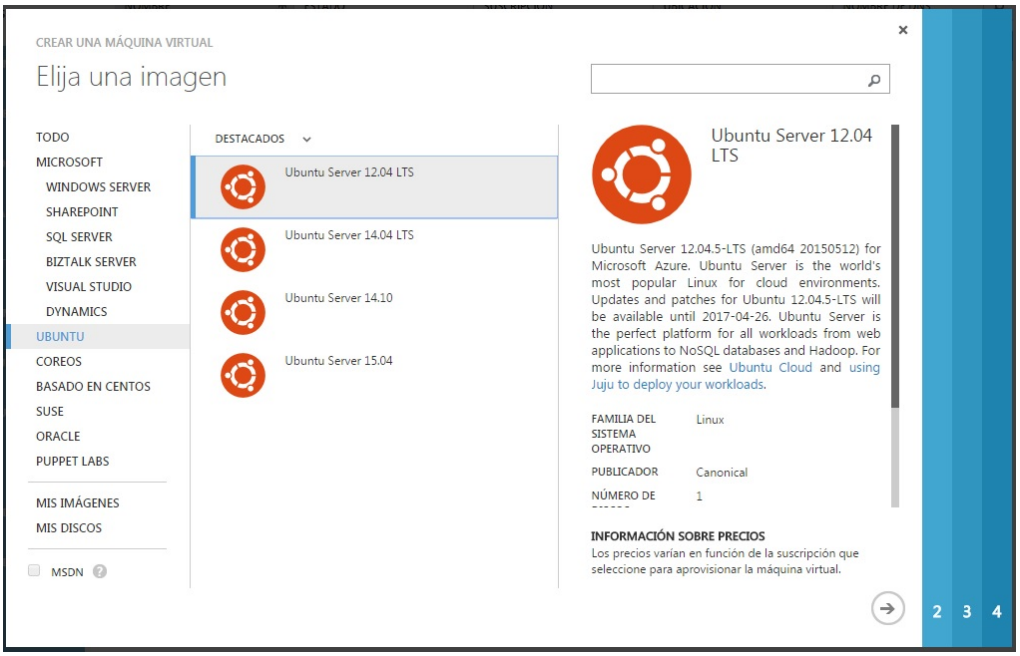


FIGURA 9.4 AZURE - SELECCIÓN DE IMAGEN DEL SISTEMA OPERATIVO

Damos "Siguiente". Ahora se nos consultan varios datos. Nos concentramos en:

- **Nombre**: Nombre que identificará a la máquina virtual. En este ejemplo: *jcc2015template*
- **Capa**: En qué nivel de suscripción se ubica: Básico o Estándar. En este ejemplo: *Básico*
- **Tamaño**: Elección de las opciones que ofrece la capa seleccionada. En este ejemplo: *A2*
- **Nuevo nombre de usuario**: Nombre de usuario para el sistema operativo. Este dato es importante ya que luego se replicará en todas las instancias que creemos. En este ejemplo usamos: *jcc2015*
- **Autenticación**: Se puede por contraseña o por ssh-key. En este ejemplo utilizamos clave: jcc2015!

FIGURA 9.5 AZURE - CONFIGURACIÓN DE MÁQUINA VIRTUAL

Presionamos “Siguiente”. Nos consultará unos últimos datos de configuración. Lo importante aquí es que seleccionemos el “***Servicio en la nube***” creado previamente: **jcc2015**

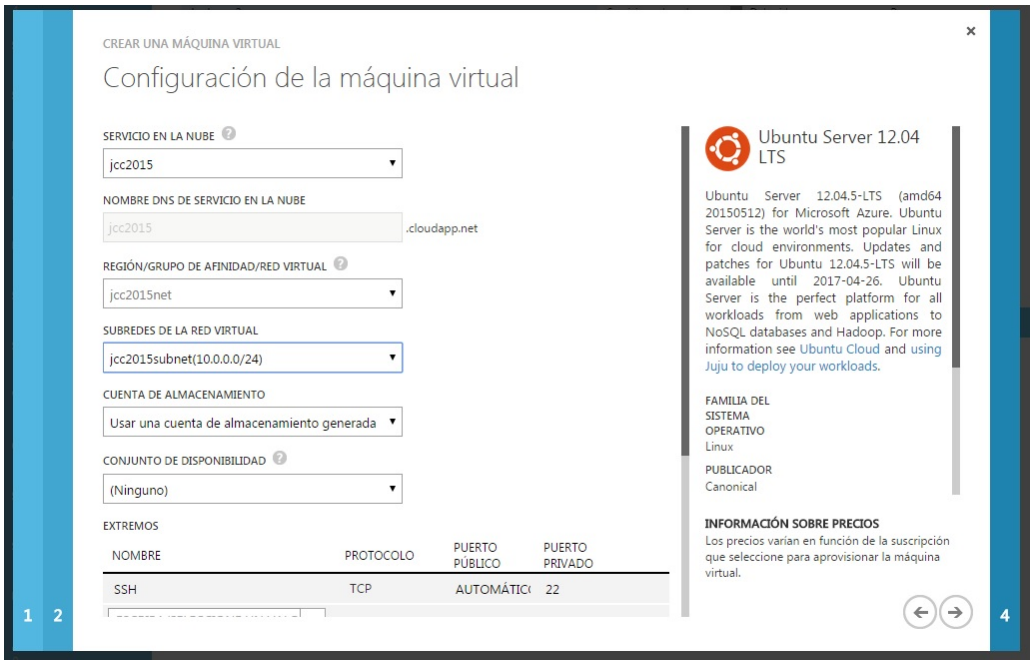


FIGURA 9.6 AZURE - CONFIGURACIÓN DE MÁQUINA VIRTUAL (1)

Presionando “Siguiente” pasamos a la última pantalla de configuración. Allí lo importante es tildar la opción que habilita la instalación del Agente de Máquina Virtual, ya que luego será utilizado.
Damos a “Finalizar”, y se iniciará la implementación del servicio con su máquina virtual.
Tener en cuenta que la máquina virtual arrancará directamente. (Generando costo)
Si consultamos el “servicio en la nube” llamado jcc2015 veremos el resumen, y que ya está iniciado:

FIGURA 9.7 AZURE - PANEL GENERAL DE SERVICIO EN LA NUBE

Con esto estamos en condiciones de probar el acceso vía SSH a la instancia creada.
Para acceder desde Windows se utiliza el cliente PuTTY [96]
Se puede acceder vía IP o con el nombre del dominio. Por ejemplo:

```
$ ssh jcc2015@jcc2015.cloudapp.net
```

Pide contraseña y accedemos:

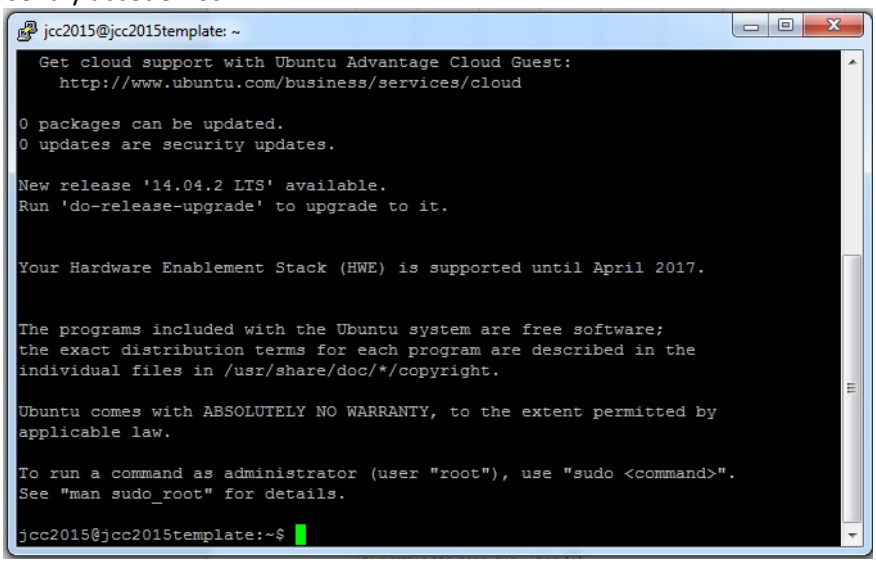


FIGURA 9.8 TERMINAL DE LA MÁQUINA VIRTUAL

A este punto ya estamos en condiciones de configurar la máquina creada para que sea utilizada como plantilla para todo el cluster.
Algunos de los pasos que vamos a realizar:

- **Instalación de Java**::

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java7-installer
$ sudo apt-get install oracle-java7-set-default
```

- **Instalación de Hadoop (usamos la versión 2.6.0):**

```
$ wget
http://apache.mirrors.tds.net/hadoop/common/hadoop-
2.6.0/hadoop-2.6.0.tar.gz


$ tar -xvzf hadoop-2.6.0.tar.gz
$ sudo mv hadoop-2.6.0 /usr/local
$ sudo mv /usr/local/hadoop-2.6.0 /usr/local/hadoop
```

Con esto queda instalado Hadoop en /usr/local/hadoop

- **Configuración de Hadoop:** Esto consiste de dos partes. Primero configurar algunas variables de entorno para que Hadoop sepa dónde ubicar los diferentes archivos que intervienen. Y por otra parte se tienen que configurar archivos específicos de Hadoop. En la primera parte se debe editar el archivo ***$HOME/.bashrc***, agregando las siguientes líneas:

```
export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop

# Native Path
export
HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/na
tive
export HADOOP_OPTS="-
Djava.library.path=$HADOOP_PREFIX/lib"

#Java path
export JAVA_HOME=/usr/lib/jvm/java-7-oracle

# Agregar el directorio "bin" de Hadoop al path
export
PATH=$PATH:$HADOOP_HOME/bin:$JAVA_PATH/bin:$HADOOP_H
OME/sbin
```

El otro archivo que interviene en este paso se encuentra en la carpeta ***$HADOOP_HOME/etc/hadoop/*** y se llama **hadoop-env.sh**. Se le debe agregar la siguiente línea para que sepa dónde ubicar la JVM: ***export JAVA_HOME=/usr/lib/jvm/java-7-oracle*** En la segunda parte de esta configuración, intervienen archivos propios de Hadoop:

- **$HADOOP_HOME/etc/hadoop/core-site.xml**: Configurar los parámetros:
  - *fs.default.name*: Con el valor "*hdfs://master:9000*". Donde "master" es el nombre del host del namenode.
  - *hadoop.tmp.dir* : Con el valor *$HOME/tmp* (reemplazar $HOME por el valor de la capeta del usuario, y crear esta carpeta)
- **$HADOOP_HOME/etc/hadoop/hdfs-site.xml**:
  - *dfs.replication*: Se configura en 2, ya que tendremos 2 nodos esclavos.
  - También se configuran las rutas del datanode y el namenode: *$HOME/hdfs/datanode* y *$HOME/hdfs/namenode* respectivamente (crear estos directorios)
- **$HADOOP_HOME/etc/hadoop/mapred-site.xml:** Se le indica el framework de MapReduce a utilizar (YARN) en el parámetro *mapreduce.framework.name*.
- **$HADOOP_HOME/etc/hadoop/yarn-site.xml:** Tiene algunos parámetros de configuración sobre el YARN. Los principales indican dónde se ubican algunos servicios(por ejemplo, se indica la dirección del master)
  - *yarn.nodemanager.aux-services: mapreduce_shuffle*
  - *yarn.nodemanager.aux-services.mapreduce.shuffle.class: org.apache.hadoop.mapred.ShuffleHandler*
  - *yarn.resourcemanager.resource-tracker.address: master:8031*
  - *yarn.resourcemanager.address: master:8032*
  - *yarn.resourcemanager.scheduler.address: master:8030*

- Configuración del archivo ***/etc/hosts***: Aquí deben quedar todas las relaciones host-IP que van a intervenir en el clúster. No es problema si no lo sabemos de antemano, ya que podremos editarlo luego.

```
10.0.0.4   master
10.0.0.5   slave01
10.0.0.6   slave02
```

- Configuración de las claves ssh: Generar la clave para que luego los nodos puedan acceder unos a otros sin necesidad de solicitar contraseña:

```
$ ssh-keygen -t rsa -P ""
```

Ya estamos en condiciones de comenzar con la creación de la imagen.
En la máquina "plantilla" que hemos configurado, ejecutar el comando:

```
$ sudo waagent -deprovision
```

Una vez ejecutado este comando, cerrar todas las sesiones que puedan existir hacia esa máquina virtual, y apagarla desde el panel de administración de Microsoft Azure.

Dentro del panel de administración de la máquina virtual, debemos seleccionar “Capturar”. Allí se solicitará un nombre y una descripción para capturar la imagen: En este ejemplo elegimos como nombre jcc2015template-2015. Tildar la opción que indica que ya corrimos el comando de waagent -deprovision. Esto se ilustra en la figura 9.9. *ACLARACION*: Una vez creada la imagen, la VM no estará más disponible.

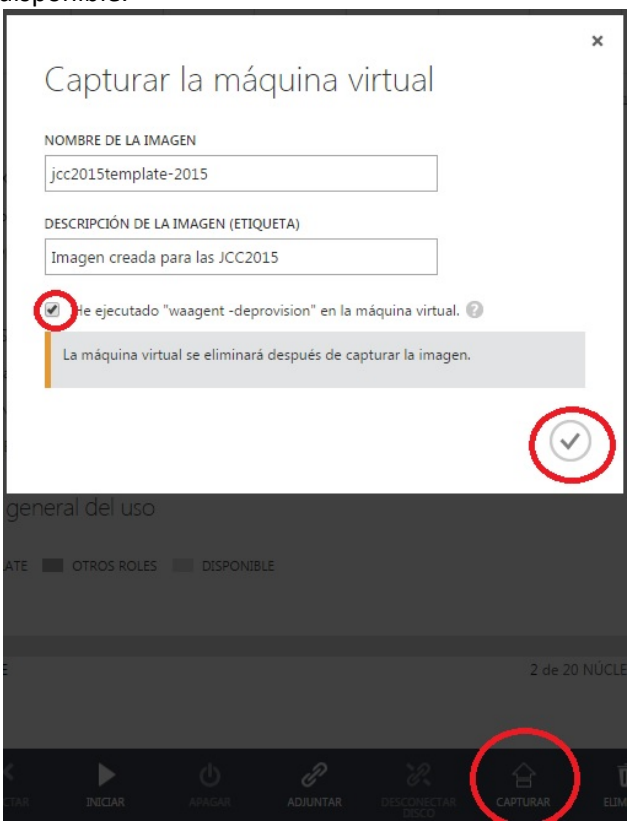


FIGURA 9.9 AZURE - CONFIGURACIÓN PARA CAPTURAR UNA MÁQUINA VIRTUAL

Una vez finalizado el proceso de creación, podemos ver la nueva imagen dentro de “Máquinas Virtuales → Imágenes”:

FIGURA 9.10 AZURE - PANEL DE "MÁQUINAS VIRTUALES"

A continuación, crearemos un clúster de computadoras utilizando la imagen creada.

Básicamente, la creación del clúster implica crear “*n*” máquinas virtuales (en este ejemplo serán 3), las cuales pertenezcan a un mismo “***servicio en la nube***” y a una misma “***red virtual***”

Comenzaremos con la primera computadora del clúster, la cual denominaremos “**master**”.

1. Ir al menú de creación de máquina virtual y seleccionar “de la galería”. Elegir la imagen creada previamente

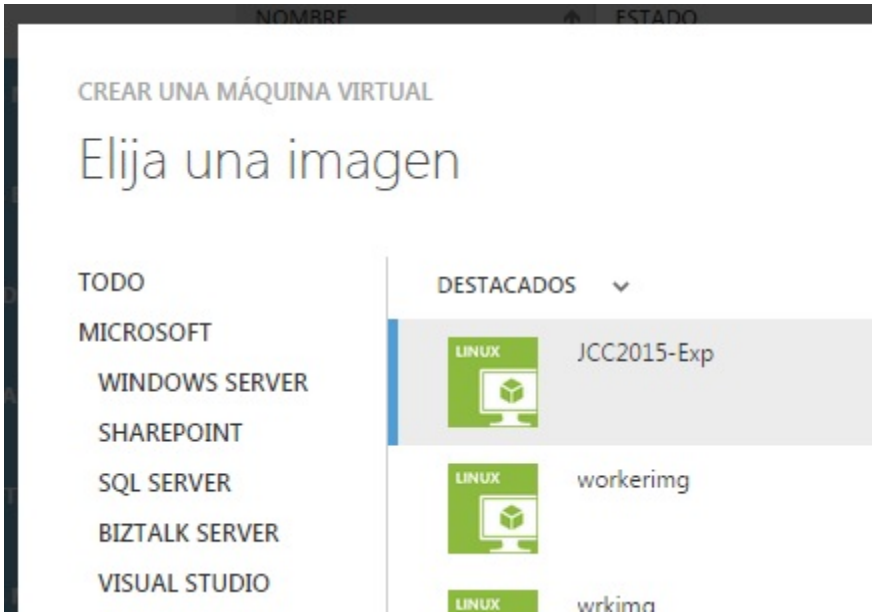


FIGURA 9.11 AZURE - SELECCIÓN DE IMÁGEN DEL SISTEMA OPERATIVO

2. Configurar parámetros de la máquina virtual como el nombre, el flavor, usuario y contraseña (figura 9.12)
    a. Nombre: master
    b. Flavor: Básico A2
    c. Usuario jcc2015
    d. Passoword: jcc2015!

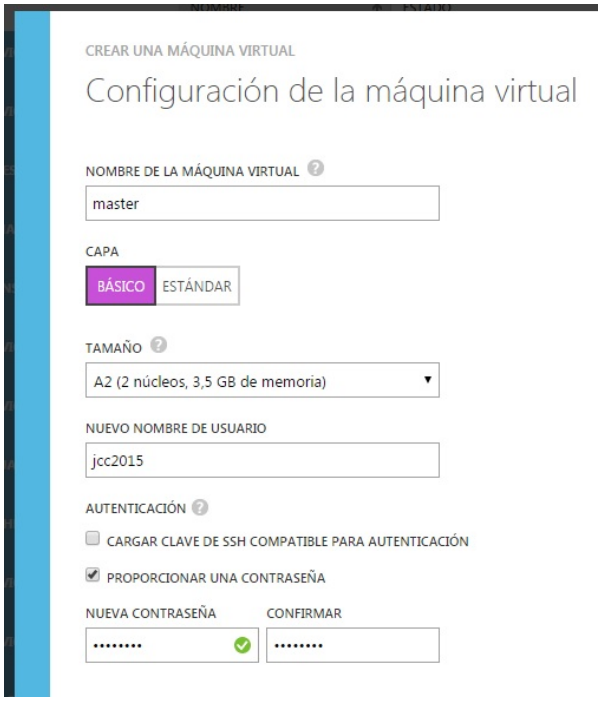


FIGURA 9.12 AZURE - CONFIGURACIÓN DE MÁQUINA VIRTUAL BÁSICA

3. Luego seleccionamos el servicio de la nube, y la subred. Al ser el master, debemos indicar también unos puertos especiales que necesitaremos para acceder a interfaces web que nos permitirán conocer el estado del clúster y el sistema de archivos. Estos puertos son: HDFS – 50070 – 50070, Clúster – 8088 – 8088, JobHistory – 19888 – 19888, quedando la configuración como sigue:

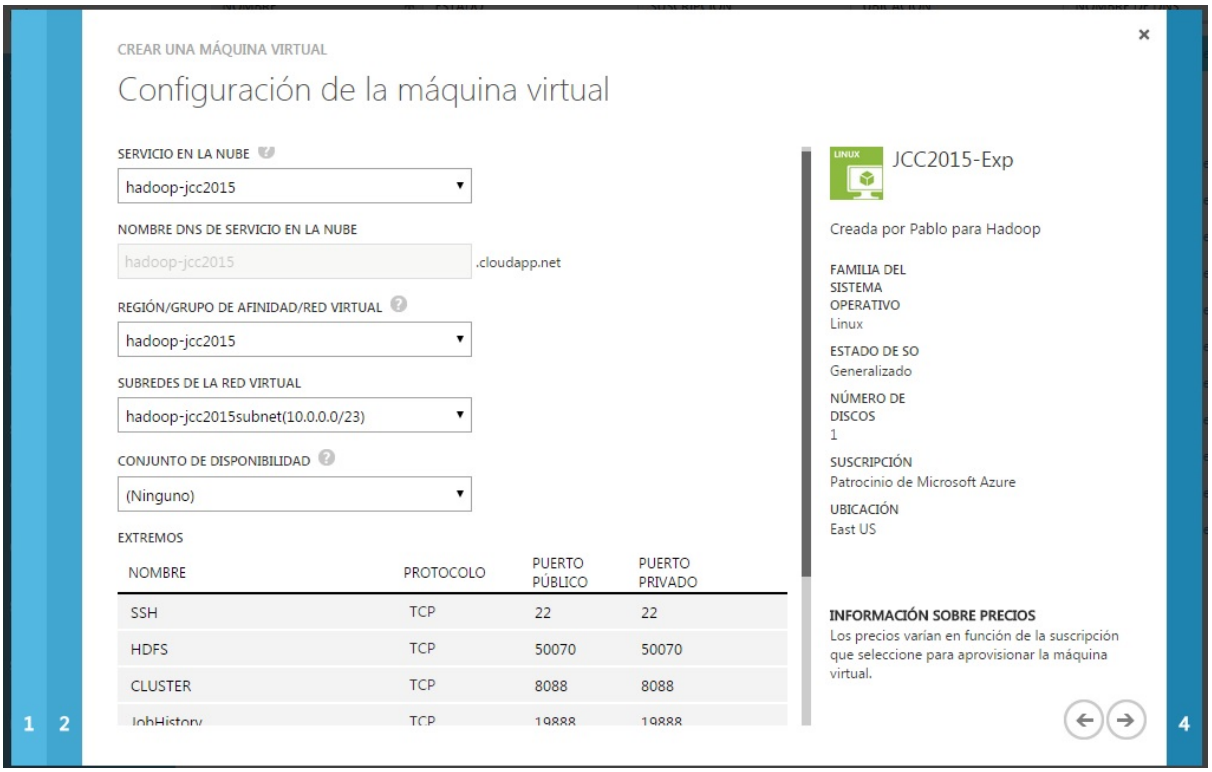


FIGURA 9.13 AZURE - CONFIGURACIÓN DE MÁQUINA VIRTUAL BÁSICA (1)

4. Una vez revisados los parámetros, podemos finalizar el procedimiento y esperar a que se cree la máquina virtual.

Para comprobar de que el "master" fue creado correctamente podemos consultar el servicio en la nube "jcc2015-hadoop" y ver su estado.

Luego debemos crear dos instancias llamadas slave01 y slave02, que son las que van a acompañar al master en este clúster Hadoop. El procedimiento es similar, sólo que en este caso no se le especifican "extremos de red" ya que no necesitan ser accedidas desde Internet. El resto de las opciones son iguales: Servicio en la nube, red, usuario, contraseña, flavor, y los demás.
Podremos verificar el estado del clúster consultando el servicio en la nube:

FIGURA 9.14 AZURE - PANEL DE SERVICIO EN LA NUBE "JCC2015"

Dentro de este panel de supervisor (figura 9.14), podremos ver los puntos de acceso disponibles desde Internet (figura 9.15). Veremos que hay 3 accesos ssh (1 por cada instancia), sumado a los 3 accesos que habilitamos en el master para poder operar:

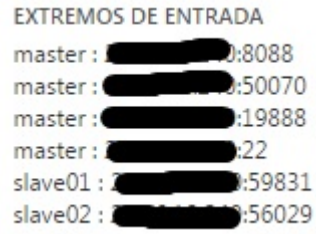


FIGURA 9.15 AZURE - EXTREMOS DE ENTRADA EN EL SERVICIO JCC2015

Ahora accedemos a las instancias a través de algún cliente SSH:

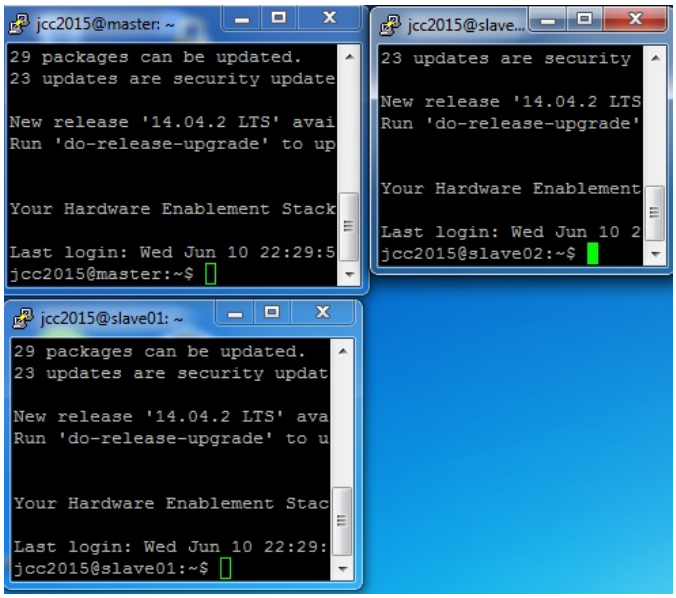


FIGURA 9.16 ACCESO A MÚLTIPLES TERMINALES

Una vez funcionando los pasos hasta este punto, estamos en condiciones de dar las últimas configuraciones necesarias previas a encender el servicio de Hadoop a nivel clúster.

En el master:

Debemos editar el archivo ***$HADOOP_HOME/etc/hadoop/slaves*** indicando los hostnames de los slaves (slave01 y slave02):

```
slave01
slave02
```

Generar y copiar la clave ssh para que los hosts se puedan ver si necesidad de tener contraseña:

```
$ ssh-keygen -t rsa -P ""
$ ssh-copy-id -i .ssh/id_rsa.pub jcc2015@slave01
$ ssh-copy-id -i .ssh/id_rsa.pub jcc2015@slave02
```

Por último, debemos formatear el NameNode, para que quede listo para ser utilizado. Este proceso también lo hacemos desde el master.

```
$ hdfs namenode -format
```

Con esto listo, estamos en condiciones de iniciar servicio de Hadoop en el clúster:

```
# Iniciar el namenode
$ hadoop-daemon.sh start namenode
# Luego ejecutar los datanodes de los slaves, mismo desde el
master
$ hadoop-daemons.sh start datanode
# Comprobar que el HDFS inició correctamente
# http://jcc2015-hadoop.cloudapp.net:50070/
# Iniciar el resource manager en el master
$ yarn-daemon.sh start resourcemanager
# Iniciar el resource manager en los slaves (desde el master)
$ yarn-daemons.sh start nodemanager
# Comprobar estado del resource manager
http://jcc2015-hadoop.cloudapp.net:8088/cluster
# Iniciar el job history en el master
$ mr-jobhistory-daemon.sh start historyserver
# Comprobar que inició bien
# http://jcc2015.cloudapp.net:19888/jobhistory
```

Para detener todos los servicios de Hadoop del clúster:

```
$ mr-jobhistory-daemon.sh stop historyserver
$ yarn-daemons.sh stop nodemanager
$ yarn-daemon.sh stop resourcemanager
$ hadoop-daemons.sh stop datanode
$ hadoop-daemon.sh stop namenode
```

Glosario

API: Interfaz de programación de aplicaciones (del inglés: Application Programming Interface)
HDFS: Sistema de archivos distribuido de Hadoop (del inglés: Hadoop Distributed File System)
HTTP: Protocolo de transferencia de hipertexto (del inglés: Hypertext Transfer Protocol)
JSON: Notación de objeto JavaScript (acrónimo del inglés: JavaScript Object Notation)
REST: Transferencia de Estado Representacional (del inglés: Representational State Transfer)
URL: Localizador de recursos uniforme (del inglés: Uniform Resource Locator)
YARN: Otro negociador de recursos (del inglés: Yet Another Resource Negotiator)
XML: Lenguaje de Marcado Extensible (del inglés: eXtensible Markup Language)
HATEOAS: Hipermedia como el motor de estado de la aplicación (del inglés: Hypermedia As The Engine Of Application State)

REFERENCIAS

[1] «Internet Live Stats,» [En línea]. Disponible: http://www.internetlivestats.com/.

[2] B. Marr, «The Awesome Ways Big Data Is Used Today To Change Our World,» 2013. [En línea]. Disponible: https://www.linkedin.com/pulse/20131113065157-64875646-the-awesome-ways-big-data-is-used-today-to-change-our-world.

[3] «Wikipedia - Business Intelligence,» [En línea]. Disponible: https://en.wikipedia.org/wiki/Business_intelligence.

[4] C. Strauch, «NoSQL Databases,» [En línea]. Disponible: http://www.christof-strauch.de/nosqldbs.pdf.

[5] J. Dean y S. Ghemawat, «MapReduce: Simplified Data Processing on Large Clusters,» 2004. [En línea]. Disponible: http://static.googleusercontent.com/media/research.google.com/es//archive/mapreduce-osdi04.pdf.

[6] R. B. Fragoso, «¿Qué es Big Data?,» 2012. [En línea]. Disponible: https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/.

[7] REST In Practice, pp. 93-97.

[8] R. T. Fielding, «Architectural Styles and the Design of Network-based Software Architectures,» 2000. [En línea]. Disponible: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[9] L. Richardson, «crummy,» [En línea]. Disponible: https://www.crummy.com/self/.

[10] J. Webber, S. Parastatidis y I. Robinson, REST In Practice, O'Reilly, pp. 5-11.

[11] J. F. Kurose y K. W. Ross, Redes de Computadores: Un enfoque Descendente Basado en Internet, p. 75.

[12] «RPC: Remote Procedure Call,» Protocol Specification, 1988. [En línea]. Disponible: https://www.ietf.org/rfc/rfc1057.txt.

[13] M. Fowler, «Richardson Maturity Model,» 2010. [En línea]. Disponible: http://martinfowler.com/articles/richardsonMaturityModel.html.

[14] «Hypertext Transfer Protocol -- HTTP/1.1,» 1999. [En línea]. Disponible: https://tools.ietf.org/html/rfc2616#section-9.

[15] « Hypertext Transfer Protocol -- HTTP/1.1,» 1999. [En línea]. Disponible: https://tools.ietf.org/html/rfc2616#section-10.

[16] «The Atom Syndication Format,» 2005. [En línea]. Disponible: https://tools.ietf.org/html/rfc4287.

[17] «Forms,» [En línea]. Disponible: https://www.w3.org/TR/html401/interact/forms.html.

[18] «HTTP Over TLS,» 2000. [En línea]. Disponible: https://tools.ietf.org/html/rfc2818.

[19] «HTTP Response Message Format,» [En línea]. Disponible: http://www.tcpipguide.com/free/t_HTTPResponseMessageFormat.htm.

[20] J. O.Kephart y D. M. Chess, «The Vision of Autonomic Computing,» [En línea]. Disponible: http://pages.cs.wisc.edu/~swift/classes/cs736-fa06/papers/autonomic-computing.pdf.

[21] P. Mell y T. Grance, «The NIST Definition of Cloud Computing,» 2011. [En línea]. Disponible: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[22] «A Break in the Clouds: Towards a Cloud Definition,» 2009. [En línea]. Disponible: http://ccr.sigcomm.org/online/files/p50-v39n1l-vaqueroA.pdf.

[23] E. Sozoñiuk, «Multitenant (o tenencia multiple). Qué es? Ventajas y desventajas. Aplicación a Weblogic Srv.,» 2016. [En línea]. Disponible: https://www.linkedin.com/pulse/multitenant-o-tenencia-multiple-qu%C3%A9-es-ventajas-y-srv-sozo%C3%B1iuk.

[24] «4 Types of Cloud Computing Deployment Model You Need to Know,» IBM developerWorks, 2015. [En línea]. Disponible: https://www.ibm.com/developerworks/community/blogs/722f6200-f4ca-4eb3-9d64-8d2b58b2d4e8/entry/4_Types_of_Cloud_Computing_Deployment_Model_You_Need_to_Know1?lang=en.

[25] «Patterns: Service-Oriented Architecture and Web Services,» IBM RedBooks, 2004. [En línea]. Disponible: http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf.

[26] «DEMODS: A Description Model,» [En línea]. Disponible: http://dsg.tuwien.ac.at/staff/truong/publications/2012/truong-aina2012-submitted.pdf.

[27] «Gartner,» [En línea]. Disponible: http://www.gartner.com/technology/about.jsp.

[28] «Azure Pricing calculator,» [En línea]. Disponible: https://azure.microsoft.com/en-us/pricing/calculator/.

[29] «Amazon Simple Monthly Calculator,» [En línea]. Disponible: http://calculator.s3.amazonaws.com/index.html.

[30] Google, «Google,» [En línea]. Disponible: http://google.com.ar.

[31] «Facebook,» [En línea]. Disponible: http://www.facebook.com.

[32] «Amazon,» [En línea]. Disponible: https://aws.amazon.com/es/).

[33] «LinkedIn Argentina,» [En línea]. Disponible: https://ar.linkedin.com/.

[34] «Netflix Argentina,» [En línea]. Disponible: https://www.netflix.com/ar/.

[35] «Spotify Argentina,» [En línea]. Disponible: https://www.spotify.com/ar/).

[36] «Analytics: The Real-world Use of Big Data,» [En línea]. Disponible: http://www.ibmbigdatahub.com/presentation/analytics-real-world-use-big-data.

[37] J. S. Ward y A. Barker, «Undefined By Data: A Survey of Big Data Definitions,» [En línea]. Disponible: https://arxiv.org/pdf/1309.5821.pdf.

[38] «An Enterprise Architect's Guide to Big Data,» Reference Architecture Overview, 2016. [En línea]. Disponible: http://www.oracle.com/technetwork/topics/entarch/articles/oea-big-data-guide-1522052.pdf.

[39] «CRM DEFINICION,» [En línea]. Disponible: http://www.crmespanol.com/crmdefinicion.htm.

[40] «Pregel: A System for Large-Scale Graph Processing,» [En línea]. Disponible: http://www.dcs.bbk.ac.uk/~dell/teaching/cc/paper/sigmod10/p135-malewicz.pdf.

[41] «MapReduce and PACT - Comparing Data Parallel Programming Models,» [En línea]. Disponible: http://stratosphere.eu/assets/papers/ComparingMapReduceAndPACTs_11.pdf.

[42] «Apache Software Foundation,» [En línea]. Disponible: http://www.apache.org/.

[43] «Apache Nutch,» [En línea]. Disponible: http://nutch.apache.org/.

[44] «Commodity Computing,» [En línea]. Disponible: http://www.vmware.com/company/news/articles/wsj_2.html.

[45] J. Gray y D. P. Siewiorek, «High Availability Computer Systems,» [En línea]. Disponible: http://research.microsoft.com/en-us/um/people/gray/papers/ieee_ha_swieorick.pdf.

[46] «Java,» [En línea]. Disponible: https://www.java.com/es/.

[47] «The 2015 Top Ten Programming Languages,» 2015. [En línea]. Disponible: http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages.

[48] «bzip2,» [En línea]. Disponible: http://www.bzip.org/.

[49] «LZ4,» [En línea]. Disponible: http://cyan4973.github.io/lz4/.

[50] «Snappy,» A fast compressor/decompressor, [En línea]. Disponible: https://google.github.io/snappy/.

[51] «zlib,» [En línea]. Disponible: http://zlib.net/.

[52] «A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS,» [En línea]. Disponible: http://www.repairfaq.org/filipg/LINK/F_crc_v3.html.

[53] «Public Key Cryptography for Initial Authentication in Kerberos (PKINIT),» 2006. [En línea]. Disponible: https://tools.ietf.org/html/rfc4556.

[54] «Master/slave (technology),» [En línea]. Disponible: https://en.wikipedia.org/wiki/Master/slave_(technology).

[55] «Hadoop Rack Awareness,» [En línea]. Disponible: https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/RackAwareness.html.

[56] «Hadoop Safemode,» [En línea]. Disponible: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html#Safemode.

[57] «Realizar una copia de seguridad de tu PC y restaurarla,» [En línea]. Disponible: http://windows.microsoft.com/es-ar/windows7/create-a-restore-point.

[58] «HDFS Federation,» Apache Hadoop, [En línea]. Disponible: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html.

[59] «Información general sobre escalabilidad,» [En línea]. Disponible: https://msdn.microsoft.com/es-ar/library/aa292203(v=vs.71).aspx.

[60] «Demonio (informática),» [En línea]. Disponible: https://es.wikipedia.org/wiki/Demonio_(inform%C3%A1tica).

[61] «ResourceManager REST API's.,» [En línea]. Disponible: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ResourceManagerRest.html.

[62] «NodeManager REST API's,» [En línea]. Disponible: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/NodeManagerRest.html.

[63] «MapReduce Application Master REST API's.,» [En línea]. Disponible: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredAppMasterRest.html.

[64] «Class FileSystem Reference,» [En línea]. Disponible: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/fs/FileSystem.html.

[65] «Class HashPartitioner<K,V> Reference,» [En línea]. Disponible: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/lib/partition/HashPartitioner.html.

[66] «Class InputSplit Reference,» [En línea]. Disponible: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/InputSplit.html.

[67] «Class RecordReader<KEYIN,VALUEIN> Reference,» [En línea]. Disponible: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/RecordReader.html.

[68] «Class RecordWriter<K,V> Reference,» [En línea]. Disponible: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/RecordWriter.html.

[69] «Amazon EC2 – Hospedaje de servidores virtuales,» [En línea]. Disponible: https://aws.amazon.com/es/ec2/.

[70] «Amazon Regions and Availability Zones,» [En línea]. Disponible: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[71] J. S. Perry, «Java language basics,» 2010. [En línea]. Disponible: https://www.ibm.com/developerworks/java/tutorials/j-introtojava1/.

[72] «Java Reserved Opcodes,» [En línea]. Disponible: http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html#jvms-6.2.

[73] «A Beginner's Guide to Integrated Development Environments,» [En línea]. Disponible: http://mashable.com/2010/10/06/ide-guide/#DK_C7d_MLuqq.

[74] «Eclipse,» [En línea]. Disponible: https://eclipse.org/.

[75] «Apache Maven,» [En línea]. Disponible: https://maven.apache.org/.

[76] «Spring Framework,» [En línea]. Disponible: https://spring.io/.

[77] «Inversion of Control Containers and the Dependency Injection pattern,» [En línea]. Disponible: http://martinfowler.com/articles/injection.html.

[78] «Spring Boot,» [En línea]. Disponible: http://projects.spring.io/spring-boot/.

[79] «Apache Tomcat,» [En línea]. Disponible: http://tomcat.apache.org/.

[80] «Eclipse Jetty,» [En línea]. Disponible: http://www.eclipse.org/jetty/.

[81] «Simple Logging Facade for Java (SLF4J),» [En línea]. Disponible: http://www.slf4j.org/.

[82] «GIT,» [En línea]. Disponible: https://git-scm.com/.

[83] «GitHub,» [En línea]. Disponible: https://github.com/.

[84] «Introducción a JSON,» [En línea]. Disponible: http://www.json.org/json-es.html.

[85] «A Universally Unique IDentifier (UUID) URN Namespace,» 2005. [En línea]. Disponible: https://www.ietf.org/rfc/rfc4122.txt.

[86] «Apache Eagle,» [En línea]. Disponible: http://eagle.incubator.apache.org/.

[87] «Azkaban,» [En línea]. Disponible: https://azkaban.github.io/.

[88] «WebHDFS REST API,» [En línea]. Disponible: https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/WebHDFS.htmlWebHDFS REST API.

[89] «HttpFs,» [En línea]. Disponible: https://hadoop.apache.org/docs/r2.4.1/hadoop-hdfs-httpfs/.

[90] «-X Command-line Options,» Oracle, [En línea]. Disponible: https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/optionX.html.

[91] «time(1) - Linux man page,» [En línea]. Disponible: https://linux.die.net/man/1/time.

[92] «cURL,» [En línea]. Disponible: https://curl.haxx.se/.

[93] «JCC&BD 2015,» Facultad de Informática, 2015. [En línea]. Disponible: http://www.jcc2015.info.unlp.edu.ar/wordpress/.

[94] «man scp,» [En línea]. Disponible: https://linux.die.net/man/1/scp.

[95] «Swagger,» [En línea]. Disponible: http://swagger.io/.

[96] «PuTTY,» [En línea]. Disponible: http://www.putty.org/.